A PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

O VI. 2ª ÉPOCA NÚMERO 59 UNA PUBLICACIÓN DE: REVISTAS PROFESIONALES S.L. 975 Ptas. • 1.280 Esc-Cont • 5,86 € (IVA incluido)



- NAVEGADOR, CORREO... -



CAPTURA DE VÍDEO CON DELPHI (III)
Funciones y estructuras para trabajar con muestras de imágenes y sonido

DESARROLLO DE APLICACIONES CON VIDEOCONFERENCIA (y III)
Proceso final del reproductor de audio y vídeo

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (II)
Creación de un componente Delphi para la transmisión de datos binarios

BBDD: DESARROLLO CLIENTE/SERVIDOR (III)
Proceso de diseño, creación y características en SQL Server y Oracle

PROGRAMACIÓN DEL REGISTRO DE WINDOWS (II)
Análisis de aspectos del Registro y cómo trabajar con la API

SOLARIS 7
Instalación, nuevas funcionalidades y características

DIRECTX 6.1 (y IV)
Trabajando con teclado, ratón, joystick... mediante DirectInput

XML (IV)
Lenguaje XSL. Trabajando con datos: Sintaxis, Patrones, Filtros...





• JDesignerPro 3.5 • LANScan Network Monitor 1.40 • Relayfax 2.0 • iServer 1.5.0
• Windows Registry Guide 1.2 • InstallConstruct 3.2 • Java Development Kit 1.2.2
Navagadores: Netscape Communicator 4.61 Internet Explorer 5.0.2314.1003



Por fin, disfrute de un servidor de Internet en su empresa

Centro Empresarial 2000

Linux2.0 POWERED

Centro Empresarial 2000, permite administrar, a través de páginas web, una intranet de forma cómoda, rápida y eficaz, sin necesidad de profundos conocimientos informáticos.

Compatible con la mayoría de las redes actuales.

El sistema incluye:

- Servidor de páginas Web
- Servidor de Correo Electrónico
- Servidor de FTP
- Nº de accesos sin límite, tanto para los usuarios de su red, como para los que se conecten en remoto (desde su domicilio, desde un portátil, colaboradores,...)
- Nº de buzones sin límite
- Creación y mantenimiento de Listas de Correo
- Alojamiento de dominio y dirección IP fija
- 25 Mb de espacio en nuestros servidores, si no quiere estar conectado 24 h. a Internet
- Estadísticas detalladas de accesos de cada cuenta, lugares visitados,... para un control total de sus usuarios
- Instalación del Sistema Operativo Linux y del Centro Empresarial 2000
- Mantenimiento remoto del sistema y Hot Line
- Sistema programable de llamadas automáticas
- Hasta 100 Mb de transmisión mensual

Coste Total: 175.000 ptas.

- Opciones adicionales: Servidor de Fax, Conexión entre diversas sedes, Formularios electrónicos, Cursos de formación, Acceso por Nodo Local, etc...





www.virtualsw.es

Cartagena 52
28028 Madrid
Tel: 91 355 76 67
Fax: 91 355 28 95

SÓLO PROGRAMADORES

Número 59
SÓLO PROGRAMADORES
es una publicación de
REVISTAS PROFESIONALES S.L.

Editor
Agustín G. Buelta
Director
Javier Amado Buiza
Coordinador Técnico
Eduardo De Riquer Frutos
Coordinadora de Redacción
Gema Romero Moreno-Manzanaro

Colaboradores
Constantino Sánchez, Juan Luis Ceada,
Jorge Delgado, Javier Sanz, Adolfo Aladro,
Javier Toledo, Esteban Amado,
Jordi Agost

Maquetación y Tratamiento de Imagen
J. Santos

Publicidad
Paloma Seidel
Tel.: (91) 304 78 46
Mariano Sánchez (Barcelona)
Tel.: (93) 322 12 38

Suscripciones
Rosa Tabares
Tel. (91) 304 87 64 Fax: (91) 327 13 03

Preimpresión
Ochoa Plus
Impresión
I.G.Pantone
Distribución
Motorpress Ibérica
Exportación
A.D.E.A.
Distribución en Argentina
Capital: Huesca y Sanabria
Interior: Beltrán

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 30-11-99
Precio en Canarias: 892 ptas. sin I.V.A.



Hechos son palabras

Este mes es nuestra intención agradaros todo lo posible. Para comenzar hemos querido mostrar a todos los seguidores de uno de los lenguajes más extendidos y compatibles, como es *Visual Basic,* cómo hacer su propia *Suite* de *Internet*. Con este artículo de portada hemos pretendido mostrar de una manera sencilla aunque no por ello con poca profundidad cómo crear una *Suite* para *Internet*. A lo largo de toda esta serie podremos comprobar cómo se desarrolla un navegador, un gestor de correo, y todo lo propio de una herramienta de estas características, eso sí, como siempre con ejemplos, código y ficheros fuentes que podéis encontrar en el CD-ROM de la revista para que no tengáis ningún tipo de problema al desarrollar vuestra propia aplicación.

Como podréis comprobar en este nuevo número los cambios comienzan a hacerse realidad, pero no todo queda ahí, como os dijimos tenemos muchas ideas y queremos agradeceros, a los miles de lectores que cada mes leéis SÓLO PROGRAMADORES, que nos hayáis ayudado y hayáis aportado vuestro granito de arena. Una vez más gracias por vuestras ideas. Pero no queremos que esto se quede aquí, por lo este mes hemos incluido una encuesta mediante la cual pretendemos hacer de SÓLO PROGRAMADORES no sólo una revista sobre programación, sino *vuestra* revista de programación, donde se traten los temas que os interesan, donde podáis encontrar todo aquello que buscáis y usarla como obra de referencia en muchas ocasiones.

También como novedad incluimos todos listados de código de la revista en el CD-ROM en formato texto para aquellos lectores que prefieran disponer de ellos, eso sí, respetando las fuentes a los que ya estamos todos acostumbrados.

Respecto al resto de temas que tratamos continuamos con la serie de "Captura de secuencias con *Delphi"*, "Programación del registro de Windows", "Transferencia de archivos punto a punto", y el "Proceso de diseño y creación de una base de datos". Decimos adiós a las series sobre "Desarrollo de aplicaciones con vídeoconferencia" y sobre "DirectX 6.1". Además los usuarios de plataformas *x86* y *Sparc* podrán saber todas las mejoras que ofrece *Sun Microsystem* con la nueva versión de *Solaris 7*, en la sección de "Herramientas", donde es analizado por nuestro grupo de técnicos. Y además la ayuda al lector, noticias, libros ... Todo esto y mucho más podéis empezar a saborearlo al pasar la página. Que disfruteis tanto como nosotros haciéndola.

Javier Amado
DIRECTOR

SÓLO PROGRAMADORES





Encuesta

Vuestra opinión, que puntualmente nos hacéis llegar a la redacción, se vuelve ahora imprescindible. Queremos mejorar y lograr que SOLO PROGRAMADORES se convierta en la revista que queréis. Gracias a vuestras respuestas en esta encuesta estamos seguros de que lo lograremos. Para premiar vuestra colaboración os tenemos preparados muchos regalos.

22

Video

CAPTURA DE SECUENCIAS CON DELPHI (III)

En las dos partes anteriores explicamos la creación del componente y cómo añadirle propiedades y métodos. En esta ocasión estudiamos algunas funciones y estructuras de datos y veremos cómo podemos trabajar con las muestras de sonido e imágenes que se van capturando.

14 Visual Basic

SUITE PARA INTERNET

Si os habéis preguntado alguna vez cómo podemos añadir *Internet* a nuestras aplicaciones estándar o qué controles necesitamos para hacerlo, aquí tenéis la respuesta. Veremos cómo incluir un navegador en nuestra aplicación y las distintas posibilidades que nos ofrece. Entramos de lleno en el mundo de *Internet*, sin modificar su tecnología pero con la intención de mejorarla.

6

Noticias

NOVEDADES

Como siempre en esta sección encontraréis las noticias más importantes de todo lo relacionado con la programación. Para que os mantengáis bien informados de lo que sucede en nuestro mundo.



CONTENIDO DEL CD-ROM

Como siempre en nuestro CD-ROM podéis encontrar las mejores novedades en programación, pero como no todo el monte es orégano queremos destacaros lo que nos parece mejor: *ActiveSkin* 2.2, *WinDriver* CE 4.1.2, *Java Development Kit* 1.2.2, *LANScan Network Monitor* 1.40 e *iServer* 1.5.0 entre otros.



Comunicaciones

DESARROLLO DE APLICACIONES CON VIDEOCONFERENCIA (y V)

Finalizamos esta serie hablando de *Netmeeting*. Un *software* de amplias posibilidades que utiliza interfaces *COM* con la incorporación de componentes dinámicos.



Windows 95/98

Programación del registro de Windows (II)

Continuamos con las funciones de la *API* de *Windows* y analizaremos algunos de los aspectos del registro para poner en práctica todas las llamadas de la *API* que vayamos viendo.



Redes

Transferencia de archivos punto a punto (II)

El diseño del *software* para una combinación punto apunto requiere de un componente para la correcta transmisión de datos binarios, a través del puerto de comunicaciones. Este es el componente *Delphi* que vamos a crear en la segunda parte de esta serie.



Nuevas tecnologías

DirectX 6.1 (y IV)

Finalizamos esta serie hablando de *DirectInput* también basado en el modelo *COM*. Este componente nos permite soportar dispositivos de entrada como el ratón, el teclado, el *joystick* ...



Bases de datos

Desarrollo cliente/servidor (III)

Llega el turno del servidor. Para ello os contamos cuáles son sus características, y el proceso de diseño y creación de una base de datos, tanto en *SQL Server* como en *Oracle*.

38 www

XML (IV). EL LENGUAJE XSL (II)

El lenguaje *XSL*, que ya introdujimos el número anterior, no es un mero puente para relacionar las fuentes de datos *XML*. Su sintaxis nos proporciona un potente método para consultas, búsquedas y filtrado de datos. Cumpliendo siempre las normas del estándar *XML*.



Herramientas

Solaris 7

Sun Microsystems pretende llegar a un mayor número de usuarios de las plataformas *x86* y *Sparc*. Para ello ha mejorado las versiones anteriores de este sistema operativa y le ha añadido funcionalidades.



Libros

Como no podemos hacernos eco de todos los libros que se publican os reseñaros los que nos han parecido más interesantes. Este mes os destacamos el dedicado a XML.



Ayuda al lector

Como siempre incluimos aquellas preguntas que nos han parecido más interesantes para que podáis resolver todas vuestras dudas.

EN OCTUBRE SEGUNDA SEMANA DEL DESARROLLADOR DE BORLAND

Database DM (Soporte de desarrollo de *Borland*) está preparando la segunda semana del desarrollador de esta compañía que se celebrará a mediados de octubre en Barcelona. Contará con la colaboración de *Computer 2000* y *Softmate*. En esta prolongación de la conferencia de *Imprise* y *Borland* ofrecida el pasado mes de agosto en Filadelfia se tratarán temas sobre novedades en productos tan interesantes como: *Delphi*, *C++ Builder*, *InterBase*, *VisiBroker* o *Midas*. La finalidad principal es ofrecer soluciones a las necesidades de directores, gestores de *IT*, desarrolladores y jefes de producto.



En la dirección **http://soporte.databasedm.es/** se puede encontrar además de información general, los temarios, hacer sugerencias, etc. Por el momento podemos destacar que el temario previsto incluye novedades sobre *CORBA* y *JAVA 2* con *JBuilder 3* entre otras muchas. Tiempo al tiempo; ya veremos lo que nos depara esta expectante Segunda Semana del Desarrollador de *Borland*.

STERING SOFTWARE PRESENTA UNA NUEVA SOLUCIÓN WEB TO HOST PARA APLICACIONES VSE

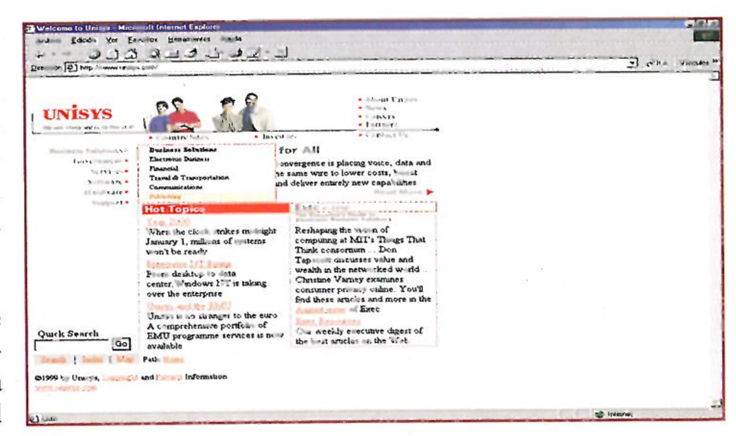
La División de Gestión de Sistemas de *Sterling Software* ha ampliado su solución *Web-to-Host VM:WebGateway*. El nuevo *VM:WebGateway CICS/VSE Interfaz* permite a las empresas ofrecer acceso *Web* a sus datos y aplicaciones. Proporcionando una interfaz gráfica más intuitiva.

Con esta nueva versión del producto los desarrolladores *VSE* pueden mejorar las capacidades *Web* utilizando *CICS/VSE COBOL* y otros lenguajes de programación *CICS*. Según una reciente encuesta más de las dos terceras partes de entre 200 *sites* prevén ofrecer acceso a *Internet* a sus aplicaciones herederas entre 6 y 24 meses.

VM:WebGateway se ofrece como una solución para llevar las aplicaciones *mainframes* a *Internet*. Dependiendo del conocimiento y preferencia, los desarrolladores de aplicaciones pueden programar *CGI* utilizando tanto *REXX*, en el caso de estar familiarizados con *VM* o *CICS/VSE COBOL* si están más acostumbrados a COBOL. La nueva interfaz permite crear *soluciones Web-to-Host* sólidas y con seguridad extremo a extremo. Si deseáis ampliar esta información *Sterling Software* posee una página *Web*, su dirección: **http://WWW.sterling.com**

UNISYS APUESTA POR MICROSOFT OFFICE 2000 COMO ELEMENTO CLAVE EN SU ESTRATEGIA DE SERVICIOS

La conocida empresa de redes, *Unisys*, opta por la integración de la archiconocida *suite* en entornos empresariales. Con esto pretende ofrecer a sus clientes nuevas soluciones ofimáticas. *Unisys* aprovechará su experiencia en sistemas informáticos distribuidos y de su infraestructura de soporte en todo el mundo para desarrollar *Office 2000* en entornos empresariales.

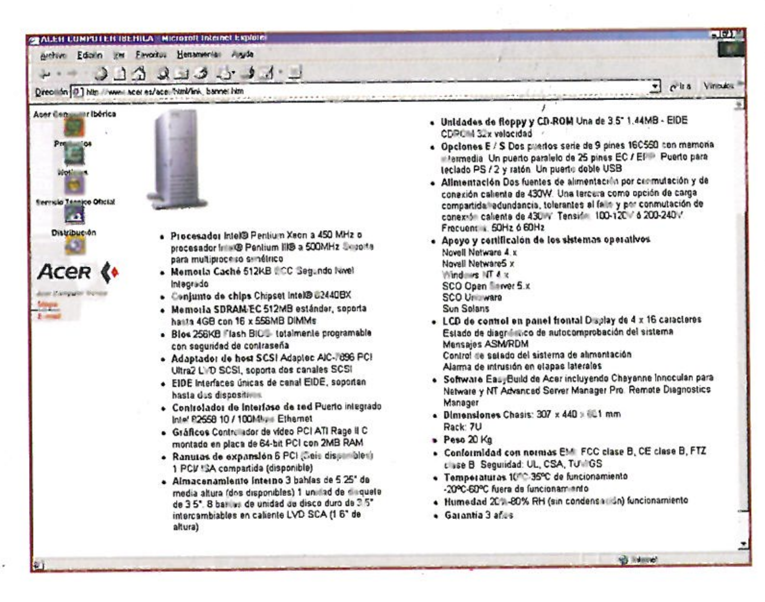


Unisys con su extensa experiencia en el diseño e implementación de entornos distribuidos multi-proveedor, ofrece la posibilidad de optimizar la infraestructura de soporte para *Office 2000* con el fin de alcanzar mayores objetivos empresariales.

Esta solución está basada en una metodología estructurada de consulta por fases, con el fin de hacer más eficaz la integración de las tecnologías de *Microsoft* en entornos heterogéneos. Para más información podéis acudir a su página web: **http://www.unisys.com**

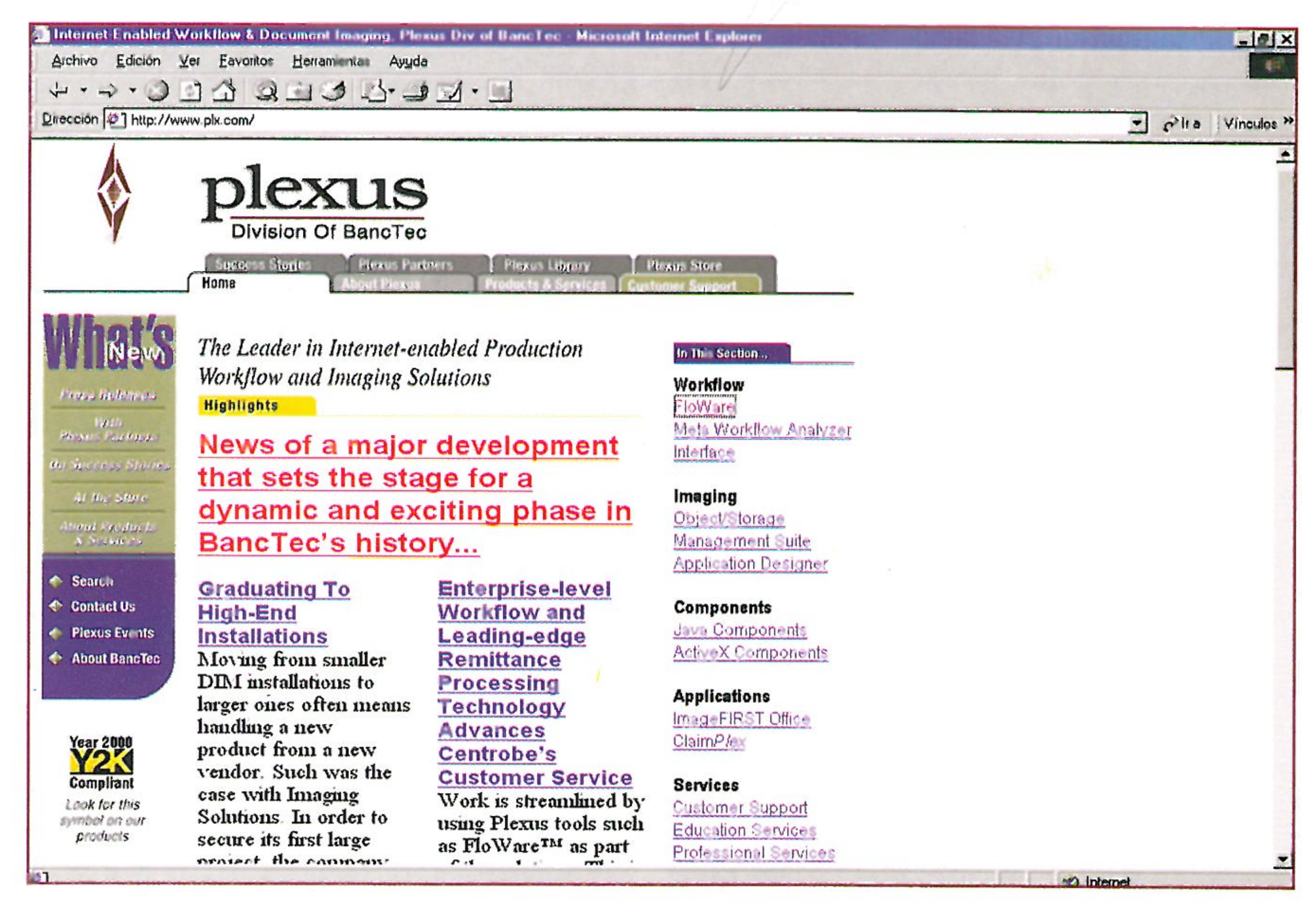
NUEVA GAMA DE SERVIDORES ACERALTOS CON PROCESADORES PENTIUM III A 550MHZ

Acer acaba de presentar su nueva gama de servidores ahora llamados **AcerAltos.** Entre los principales modelos existentes, 500, 1100, 11000 y 21000 destaca éste último por su potencia y capacidad. Sus características resultan muy interesantes para trabajar como servidor departamental en entornos *Lanchón,* una demanda de aplicaciones de misión crítica. Es capaz de disponer de hasta 4 procesadores Intel Pentium *XEON* a 450MHz o *Intel Pentium III* 500MHz con soporte para multi-proceso simétrico, con caché de segundo nivel 512KB *ECC* y bus frontal de 100MHz para alcanzar mayores tasas de transferencia de datos, con la consecuente ventaja de una mayor agilidad en las comparticiones de datos.



La placa base incorpora controlador de red *Ethernet* 100/10. El almacenamiento es mediante sistema de transferencia a dispositivo *SCSI* (integrado) con límite de 8 (*hot-swap*), balance de carga, tolerancia a fallos y unidad de energía. Soporta hasta 4 *Gbytes* de memoria *SDRAM* en 16 módulos de 556 *Mbytes DIMMs*. Dispone de la certificación para los sistemas *Novell Netware* 4.x, *Novell Netware*5.x, *Windows NT* 4.x, SCO *Open Server* 5.x, SCO Unixware y Sun Solaris Más información en: **http://www.acer.com**

BANTEC INCORPORA CORBA MIDDLEWARE SERVICE A SU LÍNEA DE PRODUCTOS BUSINESS OBJETS



CORBA services complementa su línea de entornos de desarrollo ofrecidos por *Plexus* para su *software FloWare*. Entre esta amplia línea de entornos de desarrollo se incluye *CORBA IDL, A APIs, Actives* y *Java Beans*.

Los servidores basados en *Java* ofrecen independencia de plataforma a los servidores de *FloWare*, a través de la utilización de *CORBA*. Este estándar de productos *middleware* soporta la comunicación entre sistemas distribuidos múltiples. *CORBA Services* utiliza *Interface Definitión Lenguaje*(IDL) para definir la interfaz de los objetos y a través de *CORBA ObjectRequest Broker* (ORB) se procesa la comunicación entre los objetos almacenados. Los servicios de objetos tanto para *FloWare* como para el *paquete Object / Storage Management* (O/SM) están definidos por *IDL* y configurados para que puedan ser reconocidos por *ORB*, que gestiona las interacciones entre estos servicios. Por lo que la conjunción de *IDL* de *Plexus* y *CORBA Servers Services* aumentan la funcionalidad de *FloWare* y de los servidores de archivo.

WorkfloWare es una familia de productos cliente/servidor escalable y robusta, permitiendo comenzar con una solución a redes básicas con un único servidor y permitiendo escalar hasta una instalación de producción corporativa con servidores múltiples y distribuidos. Utiliza sistemas de administración de bases de datos relacionales estándares (*RDBMS*) como medio inicial de almacenaje del trabajo en progreso, de las estadísticas y de cualquier otra información relevante. Es capaz de operar en entornos de servidores *UNIS* y *Windows NT*. Podéis ampliar estos datos acudiendo a: **http://www.banctec.com** **http://www.ramarketing.com**

HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS DE DESARROLLO

COSMO CODE 2.5
Potente entorno de trabajo para el desarrollo de aplicaciones *Java*. Incluye un gestor de proyectos, un editor de código fuente integrado, numerosas herramientas visuales, un analizador de la estructura e interacción de las clases, etc.

DEV-C++ 3.0
Entorno de desarrollo y compilador de *C* y *C++*. Posee un editor multiventana que se integra con un compilador para la construcción, enlace y ejecución de aplicaciones.

EASY CD SUITE 1.1
Genera rápidamente menús de apariencia profesional para los *CD-ROM*. Incluye los programas *Easy CD Surfer* (que crea el menú del CD) y *Easy Index Creator* (que genera la base de datos para *Easy CD Surfer*).

HOMESTEAD PUBLISHER 1.0
Editor *WYSIWYG* para páginas *Web* que soporta el procedimiento de arrastrar y soltar. Permite incluir las páginas en casi cualquier destino incluyendo *GeoCities*, *theglobe.com*, *Tripod*, *Xoom*, *homestead.com* o tu *ISP* local.

MIRACLE C COMPILER 2.2
Compilador de lenguaje *C* que se ajusta al estándar *ANSI*. Incorpora un entorno de desarrollo integrado, un compilador y un enlazador. Incluye una librería de funciones con varios programas de ejemplo.

WINDRIVER CE 4.12
Conjunto de potentes herramientas que facilitan la creación de un controlador de dispositivos *Windows*, *Windows CE* o *Linux*. Crea controladores de dispositivos basados en *PCI/ISA*, utilizando *MSDEV* Visual *C/C++*, *Borland* o *Delphi*.

LENGUAJES

VISUAL BASIC

ACTIVESKIN 2.2
ActiveX para *Microsoft Visual Basic* 5/6. Permite cambiar completamente el aspecto de nuestras aplicaciones: añadir sombras, animaciones, *morphing*, ventanas no rectangulares y efectos de semitransparencia

PROJECT ANALYZER 5.1.02
Analiza el código fuente de aplicaciones desarrolladas en *Visual Basic* 3.0, 4.0, 5.0 y 6.0 y facilita la creación de la documentación. Puede localizar y eliminar código para conseguir una reducción en el tamaño de los ficheros.

READYCODE '98
Colección de código fuente con más de 50 funciones genéricas para programadores de *Visual Basic*. Sólo hay que utilizar el procedimiento de cortar y pegar para utilizar el código correspondiente en nuestros programas.

SMPT/POP3 EMAIL LIBRARY 3.1
Conjunto de librerías que proporcionan una *API* para enviar y recibir *e-mail* a través de *Internet*. Los mensajes de correo pueden incluir ficheros atachados en formato *MIME*.

JAVA

1st Java Navigator 5.0.1.493
Utilidad que añade a los sitios *Web*, menús de navegación basados en *Java*, similares al utilizado por el explorador de *Windows*, lo que permite a los visitantes navegar por el sitio desde un menú jerárquico expandible.

DECAFE PRO 3.6
Descompilador *JAVA* que reconstruye el código fuente original. Puede descompilar los *applets Java* más complejos y ficheros binarios, generando el código fuente adecuado.

JAVA DEVELOPMENT KIT 1.2.2
Entorno de desarrollo integrado que facilita la creación de aplicaciones en lenguaje *Java*. Incorpora las nuevas versiones de *Java Virtual Machine* y librerías de clases *Java*.

JDESIGNERPRO 3.5
Facilita la creación de bases de datos interactivas en lenguaje *Java* y su inclusión en *Internet*, ya que genera automáticamente el código *Java* y *SQL*.

HTML

HOTJAVA BROWSER 3.0
Potente navegador de *Internet* desarrollado íntegramente en *Java*. Ha sido diseñado con amplias posibilidades de configuración para asegurar la inclusión de mejoras y una total adaptación a las necesidades del usuario.

JET COLOR 1.20
Interesante herramienta que facilita la selección de un código de color *HTML* para aquellos que editan manualmente su código fuente *HTML*. Permite seleccionar colores en una página *HTML* (fondo, texto, hipervínculos y enlaces visitados).

WISEBOT PRO 2.0C
Facilita la navegación por nuestros sitios creando *applets* de navegación interactiva, páginas *"What's New"* y canales *push* para *Internet Explorer* 4 o *Pointcast*.

OTROS

BUPACK 1.1
Colección de más de 230 componentes para *Delphi* 4. Incluye un visor del *Registry*, diálogos, componentes para crear aplicaciones interactivas multimedia, una herramienta que localiza rápidamente las funciones que deseemos y numerosas utilidades.

COOLCONTROLS 2.02
Genera interfaces de apariencia muy profesional para aplicaciones *Delphi*. Incluye controles para crear cajas de listas, visualizar imágenes asociadas, utilizar formas elípticas, etc.

HELP PAD 2.6
Creación de ficheros de ayuda. Facilita la creación de ficheros de ayuda para *Windows* 95 y 98. Incluye todas las herramientas para generar este tipo especial de archivos, un editor, corrector ortográfico, diccionarios, etc.

WINPOWER 1.0
Control *ActiveX* que permite ocultar nuestros programas, deshabilitar la función *Ctrl-Alt-Del* y la barra de tareas, verificar la existencia de un fichero, desconectar el ordenador, etc.

HERRAMIENTAS Y UTILIDADES

ACCESS DENIED 2.20
Programa de seguridad diseñado para activar, después de un tiempo, un *password* de acceso al equipo. Soporta hasta 45 usuarios, permite encriptar las claves de acceso y ofrece otras opciones de seguridad.

INSTALLCONSTRUCT 3.2
Herramienta de desarrollo para crear asistentes de instalación y desinstaladores de apariencia profesional, que faciliten la distribución de nuestros programas para plataformas *Windows* 3.1, 95, 98 o *NT*.

MT SYSLOG DAEMON 1.0
Analiza las actividades que se producen en el servidor. Filtra los mensajes para clasificarlos por texto, dirección *IP* o ambos. Puede enviar mensajes innecesarios u otros importantes a una carpeta separada.

POWERSTRIP 2.51.06
Optimiza el rendimiento de la tarjeta de vídeo y se ejecuta desde un icono en la barra de tareas para proporcionar un rápido acceso a los parámetros de visualización.

SPY & CAPTURE 2.60
Herramienta de espionaje para *Windows* 95, 98 y *NT*. Utiliza la posición del ratón para obtener propiedades de ventana y todos sus objetos, estilos, clases e información de procesos.

WINHEX 8.51
Editor hexadecimal que encadena, divide, analiza, compara y convierte ficheros. Posee opciones para realizar búsquedas y sustituciones, deshacer acciones y generar números.

WINXFILES 4.0
Utilidad de encriptación que incorpora un visor para trabajar con ficheros de imágenes. Soporta los formatos gráficos *BMP*, *JPG*, *GIF*, *GIF* animado (sólo en modo de previsualización), *PNG*, TIF, *PCX*, y *WMF*.

REDES

LOCALES

ESSENTIAL NETTOOLS 2.0
Software de administración de redes que incluye utilidades de análisis y seguridad. Posee un rápido escáner de *NetBIOS*, una herramienta que muestra las conexiones de la *Red*, un monitor de las conexiones externas a los recursos compartidos del ordenador, etc.

OTEX LAN 5.1
Sistema operativo para redes que cuenta con todas las características de paquetes como *Netware* y *LANtastic*. Trabaja con equipos bajo *Windows* 98, *Windows* 95, *Windows* 3.x, *DOS* y ordenadores portátiles.

REPORTMAGIC FOR WINGATE 1.0 BUILD 207
Herramienta de análisis de ficheros *LOG* para el *firewall* y servidor *proxy WinGate*. Contarán con información sobre la actividad y tráfico de este servidor.

VICOMSOFT SOFTROUTER PLUS 6.03
Programa para conectar la red *LAN* a *Internet*, redes *LAN* a *LAN* y redes *LAN* a *WAN*. Comparte el acceso a *Internet* lo que permite a múltiples usuarios de una red acceder a la *Web* utilizando una única conexión y cuenta *ISP*.

DISTRIBUIDAS

ISERVER 1.5.0
Servidor de sitios *Web* que administra páginas estáticas y dinámicas. Es

rápido, seguro, tolerante a fallos y escalable. Soporta *HTTP* 1.1, *Java Servlets*, *Common Gateway Interface* (*CGI*), *Server Side Includes* (*SSI*) y *Server Side Scripting* (*iScript*).

NEOTRACE 2.02
Utilidad de monitorización de las conexiones que muestra todos los nodos utilizados al conectar nuestro equipo con una dirección de *Internet*. Puede visualizar hasta 5 rutas a la vez y detecta los problemas.

OSTROSOFT INTERNET TOOLS 3.5
Colección de 15 utilidades integradas para las comunicaciones en *Internet* como *traceroute*, *whois*, *finger*, etc. Incluye herramientas para monitorizar *hosts*, servicios, buzones de *e-mail* y documentos *HTML*.

OTROS

LANSCAN NETWORK MONITOR 1.40
Herramienta de control de redes. Informa de la utilización de la red, los tests de conectividad y respuesta, y las actividades de *Internet*.

NSHARE FOR LAN 2.25
Permite a varios ordenadores acceder a la vez a *Internet*. Es compatible con módems *DSL*, Cable y analógicos y soporta *Microsoft Internet Exchange, Netscape, Real Audio, Electronic Mail, Telnet, FTP*, etc.

RELAYFAX 2.0
Cliente/servidor que facilita el envío de correo electrónico a un fax y viceversa. Está optimizado para trabajar con *WinGate* y *MDaemon*! Se integra con el sistema de correo electrónico existente para proporcionar a los usuarios de una red funciones de fax desde sus terminales.

DOCUMENTACIÓN /TUTORIALES

BUILD A GREAT WEB SITE ON A BUDGET 1.0
Libro electrónico interactivo que explica las cuestiones a considerar antes de publicar un sitio *Web*. Incluye temas sobre la planificación del sitio, su diseño y alojamiento en un servidor. Ofrece información sobre el dominio, los formularios *Web*, gráficos, promoción de un sitio, etc.

INTRO TO JAVASCRIPT
Completo manual de entrenamiento. Explica los fundamentos de programación a través de *JavaScript*. Incluye temas como eventos de ratón, ventanas *Pop-Up*, alertas, confirmaciones, etc. explicados con ejemplos y preguntas de revisión.

LEARN VISUAL BASIC 6 V1.5
Tutorial que facilita el aprendizaje *Visual Basic* 6. 10 lecciones (en formato *Word*) que explican desde conceptos generales de programación y fundamentos de diseño, hasta herramientas y distribución de aplicaciones de *Visual Basic*.

WEBTUTOR 3.6
Colección de tutoriales para la creación de páginas *Web*. Facilitará el aprendizaje de los principios de HTML y ayudará en los procedimientos más complejos.

WINDOWS REGISTRY GUIDE 1.2
Fichero de ayuda con los mejores trucos y consejos sobre el *Registro de Windows 95/98*. Guarda los parámetros y opciones para las versiones de 32 *bits* de *Microsoft Windows*.

IMPRESCINDIBLES

ANTIVIRUS

AntiViral Toolkit Pro 3.0.129
McAfee VirusScan 4.0.3
Panda Antivirus 6.06 Platinum

GRAFICOS

Icon Bank 4.0 Gold Edition
IconForge 4.2
MainActor 3.02
Paint Shop Pro with Animation Shop 6.0 beta 3
SureThing CD Labeler 2.0
ThumbsPlus 4.0
Xara3D 3.04

INTERNET

Añadir Pro 4.0
AutoWinNet 5.5
Copernic 99 v3.02
Cuentapasos 3.75
CuteFTP 3.0
Dial-Up Magic 1.8
Eudora Light 3.0.6
Eudora Pro 4.2
GetRight 3.34
Guardián 1.1
HomeSite 4.01
ICQ 99a beta 2.21 build 1800
MIRC 32 5.60
Net Vampire 3.3
PGP 6.0.2i
URL Organizer 2.2.4
WebTrends Enterprise Suite 3.5
WebZIP 2.75

MULTIMEDIA

AudioCatalyst 2.01
Cool Edit Pro 1.2
COWON Jet-Audio 4.6
CDH Media Wizard 3.7
Sonique 1.05
WinAmp 2.24 Complete

NAVEGADORES

Internet Explorer 5.0.2314.1003
Netscape Communicator 4.61
Opera 3.60

UTILIDADES

3DMark 99 Max b200
Adobe AcrobatReader 4.0
Advanced Registry Tracer (RegFix) 1.0a
Babylon Translator 20.12
Calendar Builder 3.2i
CDRWin 3.7e
Day Time Organizer 2000
DirectX 6.0 Castellano
DirectX 6.1 Inglés
Emergency Recovery System 8.71
SiSoft Sandra 99 v5.10
Nero 4.0.3.5
Where Is It? 2.10 beta 2
Windows Commander 4.01
WinZip 7.0

SÓLO PROGRAMADORES

ENCUESTA LECTORES Otoño 1999

¡CONSIGUE TU REGALO SEGURO!

Nos gustaría que colaboraras con nosotros para mejorar nuestra revista. Por ello, te proponemos esta encuesta. Pretendemos así conocer tu opinión para hacer de SÓLO PROGRAMADORES "tu revista".

Las 100 primeras encuentas que recibamos podrán elegir entre:

- ◆ 20 TECLADOS MICROSOFT NATURAL
- ◆ 10 ADAPTADORES WIZARD RADIO
- ◆ 70 SUSCRIPCIONES POR UN AÑO

1. Califica las secciones de Sólo Programadores (siendo 1 el valor mínimo y 10 el máximo)

	1	2	3	4	5	6	7	8	9	10
a. Portada	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
b. Editorial	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
c. Sumario	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
d. Noticias	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
e. Contenido del CD-ROM	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
f. Programación multimedia	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
g. Redes	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
h. Bases de datos	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
i. WWW	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
j. Comunicaciones	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
k. Vídeo	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
l. Windows 95/98	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
m. Herramientas	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
n. Nuevas tecnologías	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑

2. Marca con una X los temas que te interesan:

a. Programación Hardware

❑ Ensamblador ❑ Drivers ❑ Aceleradoras 3D
❑ Tarjetas gráficas ❑ Tarjetas de sonido
❑ Modems ❑ Videoconferencia Otros ....

b. Programación Básica

❑ Programación orientada a objetos ❑ Algoritmos
❑ Análisis con UML Otros ....

c. Programación Multimedia

❑ Reconocimiento de voz ❑ Juegos
❑ MPEG ❑ DirectX ❑ MP3 Otros ....

d. Internet

❑ Sockets ❑ Protocolos ❑ Servidores ❑ Seguridad
❑ Intranets ❑ HTML ❑ JavaScript ❑ CGI ❑ Servlets
❑ DTMHL ❑ XML ❑ VBScript ❑ ASP ❑ IIS ❑ Perl
Otros ....

e. Programación Windows

❑ Básica ❑ API´s ❑ Registro Otros ....

f. RAD

❑ Delphi ❑ Visual Basic ❑ Visual C++ ❑ C++ Builder
❑ JBuilder ❑ VisualAge ❑ Visual Café ❑ Visual J++
❑ Java Studio Otros ....

g. Componentes

❑ ActiveX ❑ VCL ❑ JavaBeans Otros ....

h. Sistemas Operativos

❑ Windows NT4 ❑ Windows 95/98 ❑ Windows 2000
❑ Linux ❑ Unix ❑ Novell Otros ....

i. Sistemas Distribuidos

❑ Midas ❑ COM/DCOM ❑ CORBA Otros ....

J. Bases de Datos

❑ Informix ❑ Oracle ❑ DB2 ❑ SQL Server
❑ ODBC ❑ JDBC ❑ BDE ❑ Interbase
❑ PostgreSQL Otros ....

k. Inteligencia Artificial

❑ Sistemas expertos ❑ Redes neuronales Otros ....

l. Programación Científica

❑ Representación de funciones
❑ Fórmulas matemáticas Otros ....

3. Crees que los temas tratados son:

❑ Interesantes ❑ Actuales ❑ Prácticos ❑ Aburridos
❑ Demasiado Técnicos ❑ Demasiado teóricos
❑ Mal desarrollados Otros ....

4. Piensas que se deberían incluir cursos sobre lenguajes específicos:

❑ No ❑ Si ¿Cuáles? ....

5. El número de capítulos por temas te parece:

❑ Adecuado ❑ Corto ❑ Largo Otros ....

6. Nuestro CD-ROM

a. ¿Te parece interesante? ❑Si ❑No
b. ¿Es importante incluirlo? ❑Si ❑No
c. ¿Te parece clara la interfaz? ❑Si ❑No

7. Puntua las secciones del CD-ROM (1 a 10)

	1	2	3	4	5	6	7	8	9	10
Entornos de desarrollo	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Lenguajes	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Herramientas/utilidades	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Redes	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Documentación/tutoriales	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Fuentes de los artículos	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Imprescindibles	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑

8. Cuáles de estos temas te parece importante incluir en el CD-ROM:

❑ Demos comerciales ❑ Shareware ❑ Freeware
❑ Programas de apoyo a los artículos
❑ Programas completos ❑ Programas de los lectores
❑ Navegación off-line sobre temas interés

9. Respecto a las Guías Prácticas, ¿cómo querrías adquirirlas?

a. Con la revista + incremento de precio ❑Si ❑No
b. Bajo pedido de modo individual ❑Si ❑No
c. No me resultan especialmente interesantes ❑Si ❑No

10. Eres lector de Sólo Programadores desde:

❑ Primer número ❑ Hace más de un año
❑ Unos meses ❑ Es el primer ejemplar que compro

11. ¿Cuántos ejemplares compras al año?:

❑ Soy suscriptor ❑ 1-4 ejemplares
❑ 4-8 ejemplares ❑ Más de 8 ejemplares

12. Dedicas a leer la revista:

❑ Menos de 1/2 hora ❑ 1/2 -1 hora ❑ 1-2 horas
❑ 2-3 horas ❑ Más de 3 horas
❑ Las utilizas c omo consulta

13. Además de nuestra revista compras alguna otra:

❑ No
❑ Si

Indica cuál de ellas compras habitualmente:

❑ CD Media ❑ Sólo P. Linux
❑ Linux Actual ❑ Programación Actual
❑ Sólo Linux ❑ Windows NT Magazine
❑ PC Actual ❑ Dr. Dobb´s
❑ PC World ❑ Java Developers
❑ PC Plus Otras ....

14. ¿Dispones de conexión a Internet?

❑ Si ❑ No

15. ¿Dónde utilizas habitualmente Internet?

❑ Casa ❑ Trabajo
❑ Universidad/Centro de estudios Otros ....

16. ¿Cuando has accedido por última vez a Internet?

❑ Ayer ❑ Última semana
❑ Último mes ❑ Más tiempo

17. ¿Qué número de horas mensuales dedicas a Internet?

❑ 10 horas ❑ entre 10 y 20 horas
❑ más de 30 horas

18. ¿ A qué servicios de Internet accedes?

❑ Acceso a la Web
❑ Transferencia de ficheros
❑ Correo electrónico
Otros ....

19. Indica las características de tu equipo informático:

EQUIPO: ❑ Pentium ❑ Pentium II ❑ Pentium Pro
❑ Pentium MMX ❑ Pentium III
Otros ....

MÓDEM: ❑ 14.400 Kbps ❑ 28.800 Kbps
❑ 33.600 Kbps ❑ 56.000 Kbps

SISTEMA OPERATIVO: ❑ Windows 3.X ❑ Windows 95
❑ Windows 98 ❑ Unix ❑ Windows NT ❑ Windows 2000
❑ BEOS ❑ LINUX (Indicar distribución):
❑ Debian ❑ SuSE ❑ Caldera
❑ Red Hat ❑ Esware ❑ Eurielec

20. Sugerencias ....
....

EDAD

❑ Menos de 20 ❑ De 36 a 45
❑ De 21 a 25 ❑ De 46 a 55
❑ De 26 a 35 ❑ Más de 56

NIVEL DE ESTUDIOS

❑ Primarios ❑ Titulado Medio
❑ B.U.P./F.P./ESO ❑ Titulado Superior
Indicar carrera ....
....

SITUACIÓN LABORAL

❑ Estudiante ❑ Parado
❑ Trabajo a tiempo completo
❑ Trabajo a tiempo parcial
❑ Jubilado

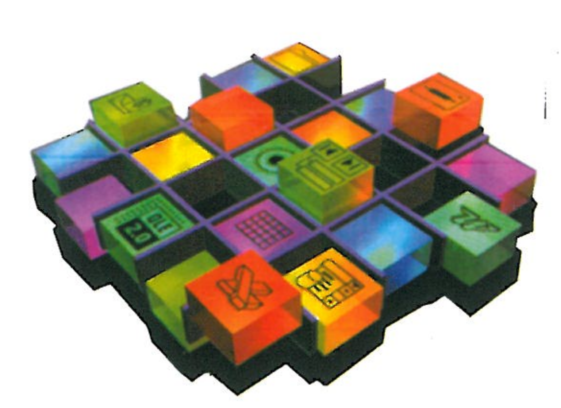
21. En tu centro de estudio se utiliza SOLO PROGRAMADORES como material de trabajo : ❑ Si ❑ No

NOMBRE Y APELLIDOS .... PROFESIÓN ....
DIRECCIÓN .... C.P. ....
POBLACIÓN .... PROVINCIA .... TELF. ....

Por si eres uno de los 100 primeros marca tu preferencia en el regalo:

❑ TECLADO MICROSOFT NATURAL ❑ ADAPTADOR WIZARD RADIO ❑ SUSCRIPCIÓN A SOLO PROGRAMADORES

POR FAVOR, ENVÍA HOY MISMO ESTE CUESTIONARIO A: SÓLO PROGRAMADORES ENCUESTA LECTORES
REVISTAS PROFESIONALES- C/ SAN SOTERO, 5, 1º PLANTA - 28037 - MADRID ● POR FAX: (91) 327 - 13 - 03



Suite para Internet (I)

Jordi Agost Moré *(agost@eup.udl.es)*

Si os habéis preguntado alguna vez cómo podemos añadir *Internet* a las aplicaciones estándar o qué controles necesitamos para hacerlo, aquí tenéis la respuesta. Con la serie que empezamos ahora veremos cómo incluir un navegador en nuestra aplicación y qué posibilidades se nos ofrecen. Pero lo haremos paso a paso, dedicando diversos capítulos a cada una de las opciones, correo, *web*, *news*....

ACCESO BÁSICO A INTERNET

Cuando en números anteriores de la revista hemos visto cómo utilizar el entorno de *Visual Basic* junto a *Internet*, ya sea mediante aplicaciones *IIS (Internet Information Server)* o páginas *ASP (Active Server Pages)*, siempre hemos tenido en mente el superar, de una forma u otra, a las páginas *HTML* estándar. Ahora vamos a entrar en el mundo de *Internet* pero sin modificar su tecnología.

CREACIÓN DE UN ACCESO DIRECTO A INTERNET

Antes de intentar arrancar o lanzar nuestro explorador en busca de un sitio *Web* concreto debemos conocer qué es un archivo *URL (Universal Resource Locator o Localizador Universal de Recursos)*, a veces también conocido con el nombre de "acceso directo a *Internet*".

Estos archivos o accesos directos a *Internet* son muy similares a los conocidos accesos directos de *Windows*, pero con la salvedad de que en vez de señalar a un determinado tipo de archivo señalan a una dirección de *Internet* o sitio *Web*.

Aunque su estructura es muy similar a la de un archivo *.INI* o a archivos de inicialización de *Windows*, no debemos confundirlos con éstos, ya que en realidad sólo contienen la *URL* de un sitio *Web*. Hemos de tener en cuenta que la estructura de los archivos de acceso directo a *Internet* es la que se detalla a continuación:

```
[InternetShortcut]
URL=HTTP://www.pruebasoloprogramado-
    res.com/
```

Un acceso directo a Internet es como un acceso directo a un archivo pero hacia una Web

Sin ir más lejos, la lista de *Favoritos* de *Microsoft Internet Explorer* consiste en una serie de archivos de este tipo. Del mismo

modo en la lista de favoritos de *Windows* (que normalmente se encuentra en el directorio *c:\windows\favoritos*) existen diversos archivos de este tipo. Vamos a crear de una forma muy simple uno de ellos con el siguiente código:

```
Open "C:\tmp.URL" For Output As #1
Print #1, "[Internet Shortcut]"
Print #1, "URL=HTTP://www.pruebasolo-
     programadores.com"
Close #1
```

Estas simples líneas crearán un acceso directo a *Internet* de nombre **Tmp.URL** y tan sólo deberemos hacer *clic* o *doble clic* sobre él para iniciar el navegador en este sitio *Web*.

EJECUTAR UN ACCESO DIRECTO A INTERNET

Ahora que hemos creado un acceso directo a *Internet* deberemos aprender cómo lanzarlo, es decir, cómo ejecutarlo desde nuestro programa. Para realizar dicha tarea podemos hacerlo mediante un par de métodos diferentes: bien llamando a la *API (Aplication Programming Interface)* o bien llamando a algún otro programa encargado de abrir el navegador con la programación correspondiente.

La llamada a la *API* de *Windows* ejecutará el programa que en ese momento se encuentre asociado con la extensión *".URL"*. Normalmente este programa coincidirá con el navegador que esté predeterminado para nuestra aplicación. Para ello necesitaremos acudir a la función de la *API ShellExecute*, cuya declaración es la siguiente:

```
Function ShellExecute Lib
     "shell32.dll" Alias
     "ShellExecuteA"
     (ByVal hwnd As Long, ByVal
     lpOperation As String,
     ByVal lpFile As String, ByVal
     lpParameters As String,
     ByVal lpDirectory As String,
     ByVal nShowCmd As Long) As Long
```

Y con dicha función y el siguiente código:

```
lRetVal = ShellExecute(lWindwos,
     "open",
                    "c:\temp.URL",
     "", "", SW_SHOWNORMAL)
```

veremos cómo se ejecuta el acceso directo a *Internet* creado.

El programa RUNDLL32.EXE es utilizado por Windows para llamar a funciones de una DLL

Aunque el uso de dicha función resulta bastante fácil, también existe un método todavía más sencillo, el que internamente utiliza *Windows* (lo podríamos comprobar en el Registro si quisiéramos) para arrancar accesos directos. Dicho método se basa en la utilización de un programa existente en *Windows* que tiene el nombre de *RUNDLL32.EXE*. Este programa, desconocido para muchos programadores, se utiliza para llamar a las funciones de una *DLL*.

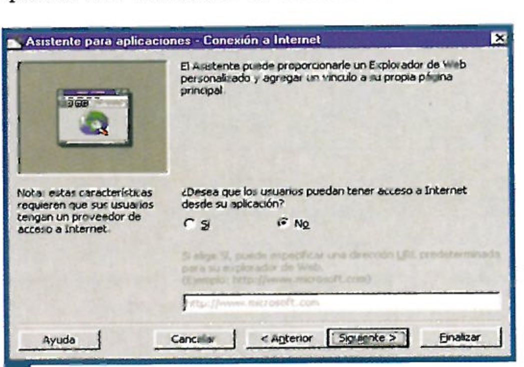


Figura 1. A través del asistente también obtendremos un navegador básico para Internet.

Si, por ejemplo, en la ventana **Ejecutar** de *Windows* (accesible a través del menú **Inicio**) ponemos la siguiente sentencia (siempre y cuando tengamos creado el fichero al que hace referencia):

```
Shell "rundll32.exe shdocvw.dll,
     OpenURL " &
          App.Path & "\temp.URL",
```

UN FORMULARIO "ACERCA DE ..." MEJORADO

Una solución muy elegante que incorporan algunos programas es un formulario **Acerca de** mejorado. Normalmente dentro de la información del programa se añade también un *link* a una determinada página de *Internet*. Nosotros lo incorporaremos con un hipervínculo.

El formulario "Acerca de ..." lo incorporaremos con hipervínculo

Para realizarlo cogeremos, por ejemplo, la página que ha podido crear nuestro proyecto por defecto y le añadiremos una etiqueta a la que nombraremos **lblURL**. Entonces crearemos un archivo temporal (que contendrá la dirección *URL*) y lo ejecutaremos tal y como hemos visto antes.

```
Private Sub lblURL_Click()

Dim nFile As Integer
nFile = FreeFile
Open App.Path & "\TMP.URL" For Output
     As #nFile
Print #nFile, "[InternetShortcut]"
Print #nFile, "URL=" & lblURL.Caption
Close #nFile

Shell "rundll32.exe
     shdocvw.dll,OpenURL " &
```

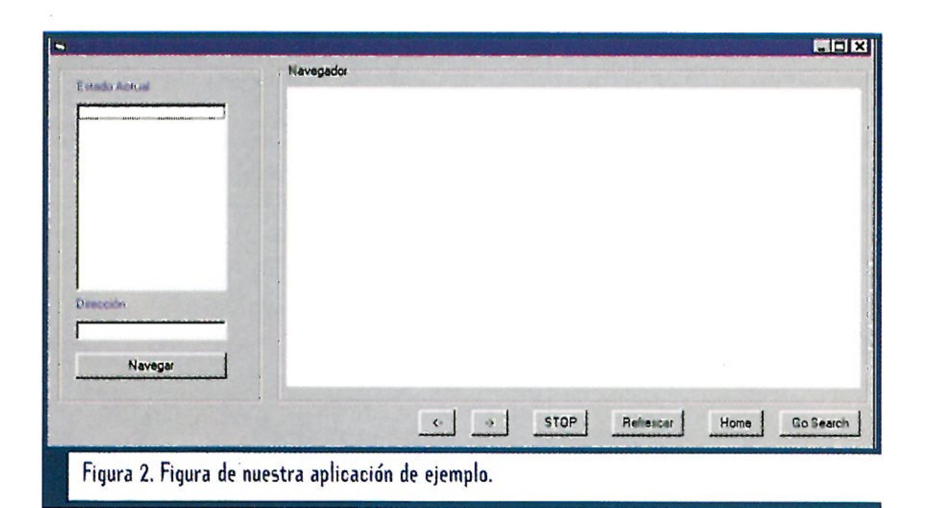


Figura 2. Figura de nuestra aplicación de ejemplo.

```
        App.Path & "\temp.URL",
     vbNormalFocus
Kill App.Path & "\TEMP.URL"
End Sub
```

AGREGAR UN NAVEGADOR A NUESTRO PROGRAMA

Algunas veces, dentro de nuestra aplicación estándar de *Visual Basic,* desearemos añadir un navegador para *Internet*, por ejemplo para añadir páginas *Web* de documentación adicional o bien páginas de soporte. Simplemente añadiendo unas pocas líneas a nuestra aplicación lograremos que ésta arranque el navegador y examine las páginas que deseemos.

La forma más sencilla de agregar el contenido de un determinado sitio *Web* a nuestra aplicación consiste en añadir el control *WebBrower*, instalado de serie en *Internet Explorer 4.0*.

Para hacer que funcione nuestro control tan sólo necesitaremos agregar el control a un proyecto *EXE* estándar (seleccionándolo de la pantalla **Componentes** dentro del menú **Proyecto** y luego activando la casilla de *Microsoft Internet Controls*). A continuación lo dibujamos en un formulario y para que funcione tan sólo deberemos escribir la siguiente línea (o una similar):

```
WebBrowser1.Navigate
     HTTP://www.prueba.com/
```

O mejor, si lo queremos hacer de una forma más limpia, insertamos una caja de texto, que tenga el nombre de **txtURL** y un botón que se llame **Command1**. La caja de texto tendrá la dirección *URL* que se cargará en nuestro navegador cuando hagamos *clic* en el botón de comando.

```
Private Sub Command1_Click()

WebBrowser1.Visible = False
WebBrowser1.Navigate txtURL.Text
WebBrowser1.Visible = True

End Sub
```

Como el control *WebBrowser* es un control que se encuentra dentro de nuestro programa de *Visual Basic*, no contiene automáticamente todas las barras de herramientas y los demás controles que encontramos asociados con *Internet Explorer*, aunque resulta sumamente fácil añadir las funcionalidades que nos faltan.

NAVEGACIÓN BÁSICA EN NUESTRO NAVEGADOR

También, en vez de navegar con el método *Navigate,* lo podemos hacer con el método *Navigate2* si lo que queremos hacer es mostrar algún directorio o carpeta en lugar de páginas *HTML* (aunque también podemos mostrarlas). Junto con el método *Navigate* o *Navigate2,* opcionalmente podemos incluir algunos *flags* para especificar información más detallada sobre la navegación, como por ejemplo el *frame HTML* de destino o las cabeceras *HTTP* enviadas al servidor.

Podemos navegar con Navigate2 en lugar de con Navigate si lo que queremos hacer es mostrar algún directorio o carpeta en vez de páginas HTML

Veamos un poco más atentamente este método. Su sintaxis es:

```
Objeto      URL [Flags,] [Frame
     Destino] [DataEnvio] [Cabeceras]
```

Como vemos el argumento *Flags* es opcional y es una constante o valor que permite añadir un determinado recurso a la lista de historial, leer o escribir en la *caché* o mostrar la nueva dirección en una nueva ventana. Su valor puede ser una combinación de los siguientes valores recogidos en la Tabla 1.

El campo *Frame* de destino también es opcional y es una cadena que indica el *frame* que tiene que

visualizarse en el navegador. Sus posibles valores son los que figuran en la Tabla 2.

El campo *DataEnvio* es opcional e indica los datos que se van a enviar durante una transacción de envío *HTTP* (por ejemplo cuando se envían datos desde un formulario *HTML* a un programa o *script*, ver gráfico).

El campo cabeceras también es opcional e indica cabeceras *HTTP* adicionales que deben ser enviadas hacia el servidor. Las utilizaremos para especificar la acción que queremos que realice el servidor, el tipo de datos que le estamos pasando o un cierto código de estado. Este parámetro, al igual que el anterior será ignorado por el método si la dirección *URL* no es una dirección *HTTP URL* (por ejemplo si estamos examinando una carpeta).

Tabla 1. Valores de Flags.

Constante	Valor	Significado
NavOpenInNewWindow	1	Abre un fichero en una nueva ventana.
NavNoHistory	2	No añade el fichero en la lista del historial. La nueva página reemplaza a la que actualmente se encuentra en la lista.
NavNoReadFromCache	4	No lee de la memoria caché durante la operación.
NavNoWriteToCache	8	No escribe en el caché los resultados de la operación.

Tabla 2. Posibles valores de Frame.

Constante/Valor	Significado
_blank	Carga el link en una ventana nueva sin nombre.
_parent	Carga el link dentro del parent del documento .
_self	Carga el documento en la misma ventana en que se hizo clic con el ratón sobre el link .
_top	Carga el link en el cuerpo de la ventana actual.
<window_name>	Un nombre de frame HTML. Si no existe dicho nombre se abrirá una nueva venta na para el link especificado.

Con los métodos GoBack y GoForward podemos navegar por la lista del historial del navegador

Continuando con los métodos de navegación diremos que si lo que queremos es ir directamente a la dirección predefinida del explorador de *Internet* o a las páginas de búsqueda utilizaremos los métodos *GoHome* o *GoSearch*. Pero si lo que queremos es mostrar la versión más actual, de la página que estamos viendo en el navegador, utilizaremos los métodos *Refresh* o bien *Refresh2*. Además con los métodos *GoBack* y *GoForward* podemos navegar por la lista del historial del navegador.

Otro evento interesante es *DownloadBegin* que se ejecuta antes de empezar la descarga de algún fichero. Por ejemplo lo podríamos utilizar para observar si en nuestro disco duro o sitio de destino hay suficiente espacio libre para guardar el fichero. Durante la descarga se irán generando diferentes eventos del tipo *ProgressChange* y después de la descarga, cuando ésta ya esté completa se generará el evento *DownloadComplete*.

WebBrowser no incluye un método de impresión que Visual Basic llame directamente

Y como punto final diremos que si utilizamos automatización *OLE* para incluir el *Microsoft Internet Explorer* en nuestro proyecto y lo que queremos es cerrar la instancia del mismo, lo que utilizaremos será el método *Quit*.

Lo que se puede ver en la Tabla 3 son algunos ejemplos de los métodos más comunes.

El control *WebBrowser* no incluye un método de impresión que pueda ser llamado directamente desde *Visual Basic*. Por lo que para imprimir el foco del programa debemos situarse en la sección que deseemos imprimir y entonces simular la pulsación de las teclas **Control + P**. Si bien esto último simplemente lo podemos lograr utilizando la siguiente sentencia:

```
SendKeys "^P"
```

Tabla 3. Métodos más comunes de navegación.

Para ir hacia el link anterior:

```
Private Sub cmdAtras_Click()
    WebBrowser1.GoBack
End Sub
```

Y para ir hacia adelante en la lista de links:

```
Private Sub cmdAdelante_Click()
  WebBrowser1.GoForward
End Sub
```

La implementación del botón de Stop sería la siguiente:

```
Private Sub Command3_Click()
  On Error Resume next
  WebBrowser1.Stop
End Sub
```

Y para refrescar:

```
Private Sub Command5_Click()
  On Error Resume Next
  WebBrowser1.Refresh
End Sub
```

Para navegar hacia la página predefinida por defecto:

```
Private Sub Command6_Click()
  On Error Resume Next
  WebBrowser1.GoHome
End Sub
```

LIMITAR LAS FUNCIONALIDADES DE NUESTRO NAVEGADOR

Vamos a realizar una aplicación muy sencilla que nos permitirá, suponiendo que en nuestra aplicación tengamos un explorador instalado, restringir el acceso a nuestro disco duro.

Lo que deseamos es que sólo podamos visualizar páginas *HTML* presentes en el dominio local o en su defecto en nuestro disco duro. Si quisiéramos encaminar al usuario a una determinada página en el momento de conectarse lo que haríamos sería lo siguiente:

```
Private Sub Form_Load()
 On Error Resume Next
 WebBrowser1.Navigate2
    "HTTP:\\www.prueba.com"
 If Err.Number <> 0 Then
   MsgBox "Error :" &
   Err.Description
 End If
End Sub
```

Y en el evento *Before Navigate2*, que se ejecuta cuando el control del navegador está a punto de navegar hacía una *URL* diferente, nos aseguraríamos de que la cadena final hacia la que quiere navegar el usuario esté especificada según nuestro patrón de permisos. De otra forma lo que haríamos sería cancelar la navegación, modificando el parámetro *Cancel* del evento *BeforeNavigate2*:

```
Private Sub
    WebBrowser1_BeforeNavigate2(ByVal
    pDisp As Object,
                    URL As Variant,
    Flags As Variant,
    TargetFrameName As Variant,
                    PostData As
    Variant, Headers As Variant,
    Cancel As Boolean)
   If (InStr(1, URL, "prueba.com") =
    0) And (InStr(1, URL, "C:") = 0)
    Then
      Cancel = True
      MsgBox "Acceso denegado por
    seguridad a la dirección: " &
    URL
   End If
End Sub
```

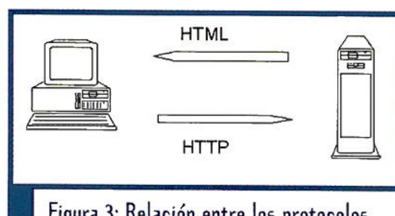


Figura 3: Relación entre los protocolos HTTP y HTML.

Cuando finaliza la navegación (suponiendo que ésta no haya sido cancelada en el *BeforeNavigate2)* se producirá el evento *NavigateComplete2*, que no incluirá el parámetro de cancelación porque ya se ha producido.

COGER EL CÓDIGO FUENTE DEL NAVEGADOR

Vamos a ver una función a la que le introduciremos el nombre de una dirección *URL* o de una dirección *Internet* y dicha función devolverá el código en formato *HTML*. Para ello al código asignado, por

ejemplo a un botón, le añadimos la siguiente sentencia:

```
Private Sub cmdVerCodigo_Click()

txtCodigoFuente =
     ObtnerCodigoFuente("c:\almanac.ht
     m")

End Sub
```

Con el control *Internet Transfer* obtenemos una implementación de dos de los protocolos más ampliamente utilizados en *Internet*: el Protocolo de Transferencia de Hipertexto (*HTTP*) y el Protocolo de Transferencia de Archivos (*FTP*).

A través del protocolo *HTTP*, podemos conectarnos con servidores de *World Wide Web* para obtener documentos *HTML*. Así en nuestro proyecto añadiremos un control *InternetTransfer* llamado **Inet1** con el fin de obtener el código fuente de la dirección especificada. Lo haremos del siguiente modo:



Figura 4. Caja de herramientas donde resalta el control Inet y el control de Microsoft Internet Explorer.

```
Public Function
     ObtenerCodigoFuente(URL As
     String) As String

ObtenerCodigoFuente =
     Inet1.OpenURL(URL)

End Function
```

PERSONALIZANDO LA INTERFAZ DE NUESTRO NAVEGADOR

El control *WebBrowser* soporta varias propiedades y eventos relacionados con la interfaz o aspecto de nuestro navegador. Vamos a ver algunas de éstas que nos permitan dar un toque más personal a nuestro control.

En primer lugar tenemos, al igual que pasa en la mayoría de controles con interfaz en el formulario, las propiedades *Height*, *Left*, *Top* y *Width*, que proporcionan la altura del control, posición horizontal respecto al contenedor, posición vertical también respecto al contenedor y la anchura del mismo.

El evento BeforeNavigate2, se ejecuta cuando el control del navegador está a punto de navegar hacia una URL diferente

Además podemos añadir algunos elementos de la interfaz del navegador a través de las propiedades *MenuBar*, que muestra o no la barra de menú, *FullScreen*, que permite ver el navegador en pantalla completa, *StatusBar* que activa o desactiva la barra de estado y *ToolBar* que muestra o no la barra de herramientas del navegador.

Dichas propiedades llevan asociadas unos eventos que son activados cuando alguna de estas propiedades cambia por cualquier motivo de valor. Los eventos son *OnMenuBar* para la barra de menús, *OnFullScreen* para la visualización completa en pantalla, *OnStatusBar* para la barra de estado y finalmente *OnToolBar* para la barra de herramientas.

AMPLIANDO LAS POSIBILIDADES

A continuación explicaremos cómo ir añadiendo posibilidades y opciones diferentes a nuestro propio programa.

CABECERA HTML EN NUESTRO PROGRAMA

Con Visual Basic también podemos hacer una pequeña subrutina que añada al principio de nuestra caja de texto enriquecido una pequeña cabecera *HTML*.

Esta cabecera puede tener utilidad, por ejemplo, si la realizamos de forma transparente al usuario, es decir, si dejamos que el usuario escriba lo que quiera y si cuando vamos a grabar le añadimos la cabecera y le cambiamos la extensión, tendremos un documento que formará la base de un futuro documento *HTML*.

```
Public Sub PasarHTML(strTitulo, txtTmp
     As RichTextBox)

Dim CRLF
```



Figura 5. Caja de herramientas donde resalta el control WebBrowser.

```
CRLF = Chr$(13) & Chr$(10)
With txtTmp

  .Text = "<!DOCTYPE>" & CRLF & CRLF
   .Text = .Text & "<HTML>" & CRLF &
      CRLF
   .Text = .Text & "<HEAD>" & CRLF &
      "<TITLE>" & strTitulo &
      "</TITLE>" & CRLF
   .Text = .Text & CRLF & CRLF
   .Text = .Text & "</HEAD>" & CRLF &
      CRLF
   .Text = .Text & "<BODY>" & CRLF &
      CRLF & "</BODY>" & CRLF & CRLF
   .Text = .Text & "</HTML>"
End With

End Sub
```

OBTENIENDO MÁS INFORMACIÓN SOBRE EL CONTROL WEBBROWSER

Podemos utilizar varias propiedades para recuperar información sobre el control *ActiveX WebBrowser*. Por ejemplo, las propiedades *LocationName* y *LocationURL* pueden ser utilizadas para obtener información sobre el sitio *Web* que actualmente estemos visualizando.

Podemos añadir algunos elementos de la interfaz del navegador a través de las propiedades MenuBar

Suponiendo que el sitio que estemos visualizando sea una página *Web*, la propiedad nos devolverá el título de esa página y *LocationURL* la *URL* de la página citada. Pero en el caso de que no visualicemos una página, sino que estemos visualizando un fichero o carpeta en nuestro ordenador o en algún ordenador de la red, entonces *LocationName* y *LocationURL* tendrán ambos el camino completo del fichero o carpeta.

Además podemos obtener información sobre el control con la propiedad *Busy*. Dicha propiedad devuelve un valor *Booleano* indicando si el control se encuentra ocupado en una operación de navegación o de descarga. Es un buen consejo consultar dicha propiedad antes de utilizar el método *Stop* para cancelar alguna operación. Veamos ahora en un ejemplo cómo utilizar esta propiedad junto al control *timer*.

```
Private Sub Timer1_Timer()

  If WebBrowser1.Busy = True Then
    lstEstadoActual.AddItem
     "Descargando..."
  Else
```




```
    WebBrowser1.Visible = True
    Timer1.Enabled = False
  End If

End Sub
```

Aquí en este ejemplo se va añadiendo a la lista que indica el estado del navegador la palabra "**Descargando**" hasta que acabemos. Y también podemos notificar cuándo acaba el documento utilizando el evento *DocumentComplete* desencadenado por el control *WebBrowser* cuando el sitio *Web* se ha abierto por completo.

```
Private Sub
     WebBrowser1_DocumentComplete(
      ByVal pDisp As Object, URL As
     Variant)

  lstEstadoActual.AddItem "Documento
     Completo"

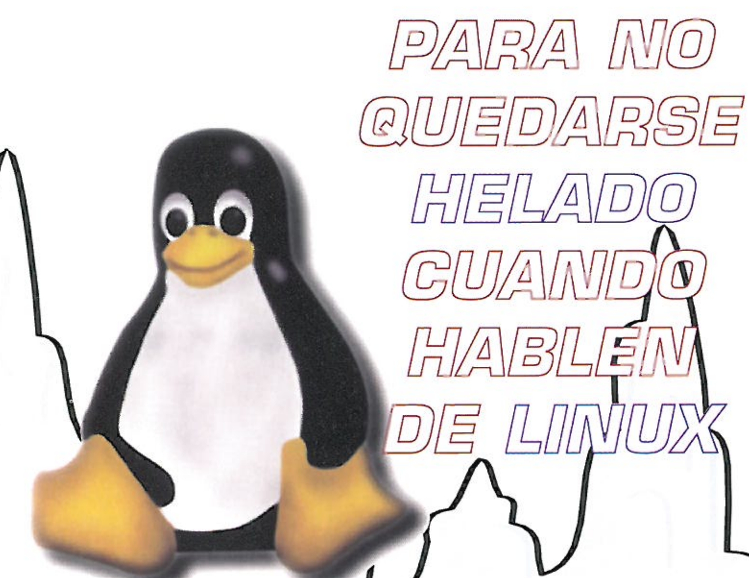
End Sub
```

Podemos saber si el navegador se encuentra en una operación de descarga o de navegación con la propiedad Busy

Hasta aquí hemos visto las posibilidades que nos ofrece el añadir un navegador a nuestra aplicación. En futuros artículos iremos completando la aplicación que hemos creado para así ir añadiendole nuevas funcionalidades del mundo de *Internet*.



QUE NO TE LÍEN CON EL <CÓDIGO>



BOLETÍN DE SUSCRIPCIÓN

40% DESCUENTO

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L. (Revista Sólo Programadores). C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revistas SOLO PROGRAMADORES y SOLO PROGRAMADORES LINUX y beneficiarme de las condiciones de estas magníficas promociónes:

❑ **SUSCRIPCION ANUAL**
24 NÚMEROS + 24 CD-ROMs
AL PRECIO DE
13.720 ptas. / 82,45€

❑ **SUSCRIPCION ANUAL ESPECIAL ESTUDIANTES**
24 NÚMEROS + 24 CD-ROMs
por sólo 10.876 ptas. / 65,36€

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❑ Domiciliación bancaria
❑ Contra reembolso

Soy antiguo suscriptor
❑ **Sí** ❑ **No**

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Eurocheque conformado con un banco español a nombre de REVISTAS PROFESIONALES, S.L

NOMBRE Y APELLIDOS:..........
EDAD:.......... PROFESIÓN:..........
TFNO:.......... DOMICILIO:..........
CIUDAD:.......... C.P:.......... PROVINCIA:..........

Promoción válida hasta agotar existencias

Datos de domiciliación:
Banco..........
Domicilio..........
Nº de Cuenta..........
Titular..........
Fecha..........
FIRMA





Captura de secuencias con Delphi (III)

Juan Luis Ceada (i5939@fie.us.es)

Esta vez veremos cómo se han implementado los eventos que proporciona el componente. Además, estudiaremos algunas funciones y estructuras de datos que serán de utilidad en el próximo artículo.

INTRODUCCIÓN

En este tercer artículo trataremos varios aspectos: explicaremos la labor de las funciones *callback*, así como los eventos que a ellas se asocian. Además, se describirán algunos aspectos referentes a la aplicación de prueba que se presentó el mes anterior, que por motivos de espacio, habíamos dejado pendientes. Por último, veremos cómo trabajar con las muestras de sonido e imágenes que se van capturando, como preludio de las aplicaciones que desarrollaremos en la próxima entrega.

EVENTOS

Una de las características más importantes y útiles que se han añadido al componente han sido una serie de eventos, que se producirán en determinadas situaciones, y que permitirán al usuario controlar cómo se está realizando el proceso de captura. En la tabla 1 se describen los eventos que se han añadido. Recordemos que los eventos relacionados con la visualización/captura de vídeo sólo se producirán cuando se esté en modo *preview*.

> Una de las características más útiles que se han añadido al componente han sido los eventos

La base de los eventos se encuentra en las funciones de *callback*. Una función de este tipo es exactamente igual que el resto de funciones (*function Nombre (...parametros...): TipoDevuelto*). Lo único que ocurre es que no se llaman desde ningún lugar de nuestro código. Es el propio *Windows* el encargado de ejecutarlas, cuando sea necesario. Existen una serie de funciones que son las encargadas de instalar las funciones de *callback* que queramos.

Veamos ahora cómo se han implementado los eventos. Nos centraremos en el evento *OnVideoStream*. El resto se implementan de forma similar, por lo que comprendiendo éste, no debe haber problemas para entender los demás.

En el listado 1, que se puede localizar en el CD-ROM se muestra todo el código necesario para realizar la correcta implementación del componente. En primer lugar, vamos a definir un mensaje, *WM_VIDEO*. Este mensaje será el que envíe la función *callback* para indicar al componente que se ha producido un evento.

A continuación tendremos que definir dos tipos de datos: *TWM_FRAMEVIDEO* y *TDatosFV*. El primero indica la estructura del mensaje a enviar. Nuestros mensajes constan de cuatro campos: el identificador del mensaje, un campo de 32 *bits* que no se usa, un puntero a la estructura de datos que queremos enviar en el mensaje y un campo resultado. El segundo tipo de datos indica la ventana de visualización con la que trabajamos y además proporciona un puntero al *frame* capturado.

En este punto se debe declarar un procedimiento tipo, *TOnFrameVideoEvent*, que será el tipo del evento *OnVideoStream*. Es decir, cuando hagamos doble *click* sobre el evento *OnVideoStream* en el *Inspector de Objetos*, *Delphi* nos invitará a que añadamos código a un procedimiento que tendrá exactamente la forma del tipo que acabamos de definir.

En la sección *private* incluimos la variable que almacena el contenido de la propiedad *OnVideoStream* (esto es, en *Delphi* los eventos se declaran como propiedades) y definimos el procedimiento *RecibirWM_VIDEO*. Este se encargará de recibir los mensajes del tipo **WM_VIDEO**. Por último, en la sección *pusblished* declaramos la propiedad *OnVideoStream*, que aparecerá en la pestaña *Events* del *Inspector de Objetos*.

La base de los eventos se encuentra en las funciones callback

Veamos ahora la implementación de las funciones *VideoCallBackProc* y *RecibirWM_VIDEO*. Le recomiendo que compagine la lectura del texto con la observación de la figura 1, para comprender más fácilmente el proceso. La primera de ellas, como su nombre indica es la función *callback*. Cada vez que llegue un *frame* durante la captura, se ejecutará. Se encarga de rellenar los campos de una variable del tipo *TDatosFV*, indicando cuál es la ventana de visualización, y en qué posición de memoria se encuentra el *frame*. Por último, envía un mensaje de tipo *WM_VIDEO*, en cuyo segundo parámetro se mandan los datos necesarios. Este mensaje es recibido por la otra función (*RecibirWM_VIDEO*). El mensaje original, podríamos decir que se "mapea" sobre una estructura del tipo *TWM_FRAMEVIDEO*, de forma que en el campo *Datos* de esta estructura tendremos la información que se envió mediante *sendmessage*.

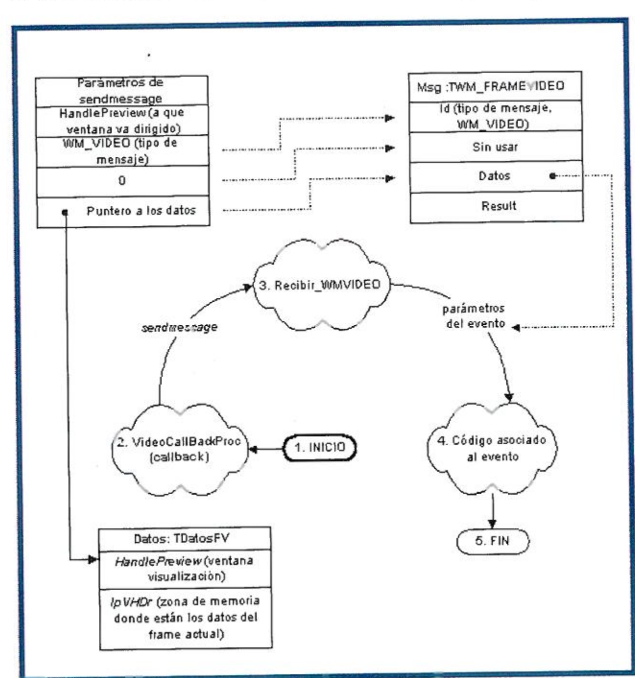


Figura 1. ¿Qué ocurre desde que se ejecuta la función callback, hasta que se ejecuta el código introducido por el usuario?

Simplemente, si no estamos en tiempo de diseño, y además el usuario ha asignado código al evento, se llama a dicho código, y se le pasan los parámetros que espera recibir. En este caso, el código asociado al evento es del tipo *TOnFrameVideoEvent*, por lo que debe recibir como parámetros el *handle* de la ventana de visualización y un puntero a una estructura de tipo *PVideoHdr*, que servirá para acceder (si se desea) al *frame*. Dicha estructura se encuentra definida en el fichero **AviCaptura.pas**.

Todos los eventos se implementan de forma similar al explicado en el texto del artículo

Por último, pero no por ello menos importante, tendremos que hacer que se instale la función *callback* (en este caso *VideoCallBackProc*) para que *Windows* sepa que existe, y que debe llamarla cada vez que llegue un *frame*. Esto se consigue mediante el procedimiento *InstalarCallBack*, el cual ejecuta la función *capSetCallbackOnVideoStream*, que recibe como parámetros el *handle* de la ventana de visualización, así como la dirección de memoria donde se encuentra nuestra función *callback*. Existe además un procedimiento *DesInstalarCallBack*, que realiza el proceso inverso. Se activarán al pasar a modo *preview* y se desactivarán al volver a modo *overlay*.

Tabla 1. Eventos añadidos al componente.

Evento	Descripción
OnFrame	Se produce antes de mostrar un *frame* (modo *preview*, visualización), lo que permite modificar el *frame* antes de visualizarlo.
OnVideoStream	Se produce antes de escribir el *frame* capturado al disco, lo que permite modificar el *frame* antes de almacenarlo. (modo *preview*, durante la captura).
OnWaveStream	Idéntico al anterior, pero con el audio.
OnError	Se produce en las siguientes situaciones: disco lleno, no se ha podido conectar la ventana de visualización al *driver*, no es posible acceder a la tarjeta de sonido, el número de *frames* perdidas supera al máximo porcentaje permitido, los *frames* no pueden ser capturados, ...
OnStatus	Se produce en las siguientes situaciones: ha finalizado la captura, se ha conseguido conectar la ventana al *driver*, se ha creado una paleta, y periódicamente, para informar del número de *frames* capturados, etc.
OnYield	Se produce al menos una vez por cada *frame* capturado. Normalmente, a este evento se le asocia un bucle de mensajes (*PeekMessage*, *TranslateMessage*, *DispatchMessage*). Es posible también filtrar y eliminar mensajes que puedan causar problemas de reentrada.
OnControl	Se llama por primera vez (*nState= CONTROLCALLBACK_PREROLL*) cuando todos los preparativos para la captura se han realizado. Una vez iniciada la captura, es llamada periódicamente (*nState=CONTROLCALLBACK_CAPTURING*)
OnClick, *OnDblClick*, *OnMouseDown*, *OnMouseMove*, *OnMouseUp*, *OnResize*	Eventos comunes a la mayoría de controles gráficos, heredados en este caso del componente base, es decir *TCustomPanel*.

Algunos eventos reciben un identificador cómo parámetro que indica la causa del suceso

Con el resto de los eventos, habría que proceder de forma muy similar. Consulte el código fuente incluido en el CD-ROM. Todos reciben como parámetro el *handle* de la ventana de visualización, como mínimo. Dependiendo del evento, se recibirán más parámetros o menos.

Un caso especial son los eventos *OnStatus* y *OnError*, que reciben un identificador que indica por qué se ha producido el suceso. En el caso del evento *OnStatus* usaremos este identificador para saber cuándo ha finalizado la captura (entre otras cosas). Esto será de utilidad para volver al modo *overlay*, después de haber realizado una captura. Sólo resta decir que en el fichero **AviCaptura.pas** se encuentran definidas una serie de constantes, que equivalen a los diferentes identificadores de error y estado que existen.

LA FUNCIÓN GETGRABANDO

En la pasada entrega dejamos pendiente la explicación correspondiente a la manera en que se actualizaba la variable **FGrabando** (cuyo valor se encargaba de devolver la función *Getgrabando*). Ya comentamos que estaba relacionado con las funciones *callback*, más concretamente con *StatusCallBackProc*.

Esta función *callback* se ejecuta periódicamente y ofrece como parámetro un identificador, que indica qué es lo último que ha ocurrido en el sistema. De este identificador nos aprovecharemos para llevar a cabo nuestros propósitos. Cuando el identificador sea *IDS_CAP_BEGIN*, querrá decir que acaba de dar comienzo la captura de vídeo, por lo que tecleamos **FGrabando = True.** Cuando el identificador es igual a *IDS_CAP_END*, escribimos **FGrabando = False.** De esta forma sencilla y rápida, podemos saber si el componente se encuentra en estado de grabación o no, sin necesidad de hacer llamadas a la función *capGetStatus*, así ganamos en velocidad y seguridad, ya que *capGetStatus* podría fallar por algún motivo (habría que estar consultando el código de error que devuelve), mientras que esta solución descarta los fallos . En el listado 2, incluido en el CD-ROM se puede observar el código de las funciones *GetGrabando* y *StatusCallBackProc*.

LA APLICACIÓN CON EVENTOS

En el artículo anterior se creó una aplicación que mostraba la mayor parte de las posibilidades del

componente. Como comentamos en su momento, había controles en la aplicación que estaban relacionados con los eventos (como es el caso de los controles *Spin*, *TImage* y *Memo* de la parte superior). Ahora vamos a explicar para qué sirve todo el código que asignamos a cada evento. La aplicación se muestra en la figura 2.

Por cierto, aunque en el primer artículo de la serie se comentó que la captura de imágenes para el artículo hubo que hacerlas en modo *preview*, ante la imposibilidad de hacerlas estando la aplicación en modo *overlay*, esto no es del todo cierto. Depende de las posibilidades de la tarjeta *SVGA* que esté presente en el equipo. Con algunas *S3* no es posible, no siendo así con las *ATI* o *Voodoo Banshee*, por poner un ejemplo. Prueba de ello es que la captura de la figura 2 se ha realizado en modo *overlay*, en un equipo dotado de una tarjeta *ATI 3D Charger*

Con respecto a los componentes de tipo *Spin* se ha incluido uno por cada evento. Esto es, cuando se produce un suceso, el código asociado a él se limita a incrementar el valor del control correspondiente. Además, en algunos eventos se ha añadido algo más de código (que puede ver en el listado 3):

Desde que se envía la orden de fin de captura, hasta que ésta finaliza, puede pasar algún tiempo

- *OnStatus*: todos los mensajes de estado que se reciben se escriben en el control *Memo*. Además, si la captura ha finalizado, volvemos a modo *overlay*. Aunque se podría pensar en incluir el paso a modo *overlay* dentro del procedimiento *StopVideo*, no resulta aconsejable. El motivo es que es muy posible que no se efectúe el cambio de un modo de visualización a otro. Cuando se da la orden de finalizar la captura, ésta no acaba inmediatamente ya que se necesita un pequeño tiempo extra para permitir que se graben los *frames* que quedan en los *buffers*. Si la llamada a *ActivarOverlay* se hace inmediatamente después de la orden de fin de captura, muy posiblemente no se ejecute (la función *capOverlay* no hace nada si la captura aún no ha finalizado).

- *OnYield*: se ejecuta un *Application.ProccessMessages*.
- *OnVideoStream*: muestra una etiqueta en la parte superior izquierda que indica que hay una grabación en proceso. Muy útil, ya que aunque un programador sabe perfectamente cuando se está grabando un vídeo (ronroneo del disco duro, saltos en la visualización, pixelado...), los usuarios no suelen tener en cuenta estos detalles. Ya que la grabación de vídeo indiscriminada puede saturar con facilidad el disco duro, es conveniente incluir algo que indique al usuario que la grabación se está llevando a cabo.

Para evitar la aparición de mensajes molestos, es necesario desactivar el evento OnControl momentáneamente

- **OnControl:** si el espacio libre en disco está por debajo de los **50** *Mb*, se detiene la captura. Nótese que una vez que se detecta que el espacio en disco está por debajo de lo normal, se desactiva el evento. Como comentamos antes, al parar la grabación, son necesarios unos segundos extras para vaciar los *buffers*. Si se detecta que falta espacio, se para la grabación, y se muestra un mensaje. Pero si quedan cuatro *frames* por grabar, por cada uno de ellos se producirá un evento de control, lo que provocaría que apareciese cuatro veces el



Figura 2. aplicación ConEventos.exe en funcionamiento.

SÓLO PROGRAMADORES

mensaje, lo cual no deja de ser molesto. El evento se vuelve a activar al pulsar sobre el botón **Grabar vídeo**

- *OnFrame*: toma el *frame* actual y lo copia en el componente *TImage* situado en la esquina superior izquierda. Para ello, toma los datos del campo *lpData* de la estructura *PVIDEOHDR*, lo transforma en un *Bitmap* y lo asigna al *TImage*. Todo el proceso es realizado por la función *SetDIBits y* será explicado en profundidad más adelante.

FUNCION SETDIBITS

Mediante la función *SetDIBits* (perteneciente a la *API* de *Windows*), hemos podido copiar las imágenes que se reciben a un control *TImage*. En definitiva, estamos trabajando con los datos que definen las imágenes recibidas. Aquí simplemente, mediante *SetDIBits* se toman los datos en formato *DIB* y se pasan al *TImage*, pero podría haberse realizado cualquier operación con ellos (como veremos en la próxima entrega). Para acceder a los datos tendremos que familiarizarnos con la estructura *TVIDEOHDR*.

Es de vital importancia familiarizarse con la estructura TVIDEOHDR

Los parámetros que recibe la función *SetDIBits* son, en orden de aparición: *handle* al *device context* de destino, *handle* al *bitmap* destino, línea a partir de la cual se comienzan a copiar datos, línea de fin (normalmente **0** y *Heigth*, respectivamente), puntero a los datos, variable de tipo *TBITMAPINFO,* y sistema de color usado (lo normal es *RGB*). Para más información, consulte la ayuda de la *API Win32* proporcionada con *Delphi.*

ESTRUCTURA TVIDEOHDR

Se encuentra definida, como no, en el fichero **AviCaptura.pas.** Su estructura se puede observar en la tabla 2.

Los campos más importantes de la estructura son *lpData* y *dwBytesUsed*. Estos campos los usaremos a la hora de acceder a la información de la imagen. Las imágenes se encuentran en formato *DIB (Device IndependentBitmap).*

Para obtener detalles del *DIB*, hay que usar la función *capGetVideoFormat()* (incluida en **AviCaptura.pas**). En la aplicación *ConEventos*, esto se hace en dos sitios: al pulsar en el botón **Conectar** y al pulsar en el botón **Formato.**

Esta función devuelve una estructura de tipo *BITMAPINFO*, que entre otras cosas, indicará el tamaño de la imagen, el número de colores, etc. Esta estructura, así como otra de tipo *BITMAPINFOHEADER*, de la cual depende, se muestra en las tabla 3. De nuevo remito al lector a la ayuda contenida en el fichero **Win32.hlp** para una información más detallada.

Los campos más importantes de TVIDEOHDR son lpData y dwBytesUsed

Mediante el siguiente código, obtenemos en la variable **BmpHead** (usada por *SetDIBits*) el tamaño, profundidad de color, etc. de las imágenes recibidas:

```
CapGetVideoFormat(
  JLCvideo.handlepreview,
  LongInt(@BmpHead),
      sizeof(BmpHead));

Imagen.picture.bitmap.width:=
   BmpHead.bmiHeader.biWidth;
Imagen.picture.bitmap.height:=
   BmpHead.bmiHeader.biHeight;
```

La primera línea se encarga de obtener la información necesaria. Las dos siguientes se utilizan para definir las dimensiones del *bitmap* del control *TImage* (no importa que el *bitmap* sea mayor que el *TImage*, ya que para eso tiene la propiedad *Stretch* a **True**). Normalmente en nuestras aplicaciones no tendremos que modificar las variables de tipo *TBITMA-*

Tabla 2. estructura TVIDEOHDR

Campo	Descripción
lpData : pBYTE;	*Buffer* que contiene los datos de la imagen
dwBufferLength : DWORD	Longitud del *buffer*
dwBytesUsed : DWORD	Nº de *bytes* con información dentro del *buffer*
dwTimeCaptured : DWORD	Milisegundos de captura que llevamos
dwUser : DWORD	Reservado
dwFlags : DWORD	Reservado
dwReserved : array[0..3] of DWORD	Reservado



PINFO. Simplemente tendremos que asignarles valor (mediante el código mostrado con anterioridad), para luego pasárselas como parámetro a alguna de las funciones de la *API* de *Windows* relacionadas con el tratamiento de imágenes.

ESTRUCTURAS RELACIONADAS CON EL SONIDO

Tan importante (o más) que poder tratar las imágenes, es poder trabajar con el sonido. Podemos acceder a los datos de audio capturados mediante un puntero, incluido en la estructura *TWAVEHDR* que recibe como parámetro el procedimiento *JLCVideoWaveStream* (al añadir código al evento *OnWaveStream*). Dicha estructura es análoga a *TVIDEOHDR*.

Lo único que se debe tener en cuenta es que los datos de audio llegan sin formato, por lo que, en la mayoría de los casos, tendremos que convertirlos a un formato estándar que *Windows* pueda entender. Supongamos que queremos grabar sonido en formato *WAV*. Para ello, habrá que crear una cabecera para el fichero, de forma que el fichero de audio generado pueda ser interpretado por la función *PlaySound* (en caso de que creemos una aplicación para reproducir el sonido), o por cualquier aplicación (como el reproductor de sonido que *Windows* proporciona).

La cabecera se genera en dos partes. La primera, que podríamos llamar parte fija, se suele generar una vez por cada ejecución de la aplicación capturadora (por ejemplo, al pulsar sobre un hipotético botón *Iniciar)*, y habrá que actualizarla cada vez que cambiemos el formato del audio.

Antes de poder enviar un fichero WAV, hay que generar una cabecera de datos

Por ejemplo, y como adelanto del siguiente artículo, en la aplicación de videoconferencia usaremos la función *capGetAudioFormat* para obtener una estructura del tipo *TWAVEFORMATEX* (ver tabla 4). Modificaremos los campos pertinentes (en este caso, *nSamplesPerSec* y *nAvgBytesPerSec*, para que se capture con una frecuencia de 8 *Khz*) y volveremos a enviar la estructura a la tarjeta sintonizadora (mediante *capSetAudioFormat*, para que a partir de entonces, capture audio con la frecuencia que le hemos indicado). Rellenamos los campos fijos de la cabecera, a partir de la información contenida en esta estructura. En la tabla 5 se muestra el formato de la cabecera de un fichero *WAV*.

La segunda parte de la cabecera, se rellena cada vez que se recibe un bloque de datos conteniendo audio (esto es, se genera cada vez que se produce un evento *OnWaveStream*). Aquí completamos los campos relativos al tamaño de los datos y del fichero (que pueden variar entre dos sucesos consecutivos, y por esto hay que estar constantemente

Tabla 3: Estructura BITMAPINFO y BITMAPINFOHEADER

Campo (BITMAPINFO)	**Descripción**
BmiHeader: TBITMAPINFOHEADER	Apunta a una estructura *BITMAPINFOHEADER*
bmiColors: TRGBQUAD	Paleta de colores de la imagen
Campo (BITMAPINFOHEADER)	**Descripción**
biSize: DWORD	Número de *bytes* requeridos por la estructura
biWidth: Integer	Ancho de la imagen, en *pixeles*
biHeight: Integer	Alto de la imagen, en *pixeles*
biPlanes: Word	Este valor debe ser 1
biBitCount: biBitCount	Número de bits por pixel (1, 4, 8, 16, 24, o 32.)
biCompression: DWORD	Tipo de compresión: *BI_RGB* (sin comprimir), *BI_RLE8* (codificación *RLE* con 8 *bit* por *pixel*), *BI_RLE4* (igual pero con 4 *bits* por *pixel*)
biSizeImage: Integer	Tamaño de la imagen en bytes (si *biCompre sion = BI_RGB*, debe valer 0)
biXPelsPerMeter: Integer	Especifica la resolución horizontal, en pixeles por metro, del dispositivo destino donde va a ser alojado el *bitmap*
biYPelsPerMeter: Integer	Especifica la resolución vertical, en *pixeles* por metro, del dispositivo destino donde va a ser alojado el *bitmap*
biClrUsed: DWORD	Indica el número de índices de color de la tabla de colores que son usados por el *bitmap*.
biClrImportant: DWORD	Indica el número de índices de color que son considerados importantes para la visualización del *bitmap*. Si es 0, todos los colores son importantes.

Tabla 4. Estructura TWAVEFORMATEX

Campo	**Tipo**	**Descripción**
wFormatTag	*WORD*	Indica el formato del audio (normalmente, *PCM*)
nChannels	*WORD*	Un valor de 1 indica mono, 2 estéreo
nSamplesPerSec	*DWORD*	Frecuencia de muestreo (8, 11.025, 22.050, 44.1 *Khz*)
NAvgBytesPerSec	*DWORD*	*nSamplesPerSec* * *nBlockAlign* (para formato *PCM*)
nBlockAlign	*WORD*	Unidad mínima de datos para el formato indicado en *wFormatTag*. Para *PCM*, es igual a (*nChannels* * *wBitsPerSample*) *div* 8.
wBitsPerSample	*WORD*	*Bits* por muestra (8 o 16 *bits*)
cbSize	*WORD*	Tamaño, en *bytes*, de la información extra que sigue a esta estructura. No se usa en el caso de formato *PCM*.

Tabla 5. Formato de la cabecera de un fichero WAV

Campo	**Tipo**	**Descripción**
Primero	*array[0..3] of char*	Contiene la cadena '*RIFF*'
Tamaño	*Integer*	Tamaño del fichero (datos + cabecera)
Segundo	*array[0..3] of char*	Contiene la cadena '*WAVE*'
Tercero	*array[0..3] of char*	Contiene la cadena '*fmt_*' (_ = espacio)
TamCab	*Integer*	Tamaño de la cabecera hasta este punto: 16
FormatTag	*Word*	Igual que *TWaveFormatEx.wFormatTag*
Channels	*Word*	Igual que *TWaveFormatEx.nChannels*
SRate	*Integer*	Igual que *TWaveFormatEx.nSamplesPerSec*
Bseg	*Integer*	Igual que *TWaveFormatEx.nAvgBytesPerSec*
BAling	*Word*	Igual que *TWaveFormatEx.nblockAlign*
BitSample	*Word*	Igual que *TWaveFormatEx.wBitsPerSample*
Cuart5	*array[0..3] of char*	Contiene la cadena '*DATA*'
NumBytes	*Integer*	Tamaño de los datos en *bytes*

actualizando la cabecera). Una vez completada la cabecera, crearíamos un *stream*, cuyo contenido sería la cabecera al comienzo, seguido de los datos. Más o menos, una cosa así:

```
Fichero WAV = Cabecera +
    PWaveHdr.lpData^
```

(donde *PWaveHDr* es un puntero a *TWAVEHDR*, que será, ni más ni menos, que lo que reciba el procedimiento *JLCWaveStream*). Después, sólo quedaría grabar el bloque de datos a un fichero, para obtener nuestro preciado archivo *WAV*.



CONCLUSIÓN

En la entrega anterior indicamos que ibamos a desarrollar una pequeña aplicación de Videoconferencia. Aunque no lo parezca, ya hemos comenzado a trabajar en ella. Hemos explicado bastantes aspectos teóricos que nos servirán para desarrollar esta aplicación, así como otra adicional: un detector de movimiento. Durante el desarrollo de las dos, haremos constantes referencias a lo que hemos aprendido en este artículo. En definitiva, en la próxima entrega la diversión será mucho mayor. Podéis ver un adelanto de una de las aplicaciones en la figura 3.

En el CD-ROM, podreís encontrar de nuevo el componente de vídeo y la aplicación de ejemplo *ConEventos*. El componente no ha variado con respecto al distribuido el pasado mes, por lo que aquellos que ya lo tengan, no necesitan instalarlo de nuevo.

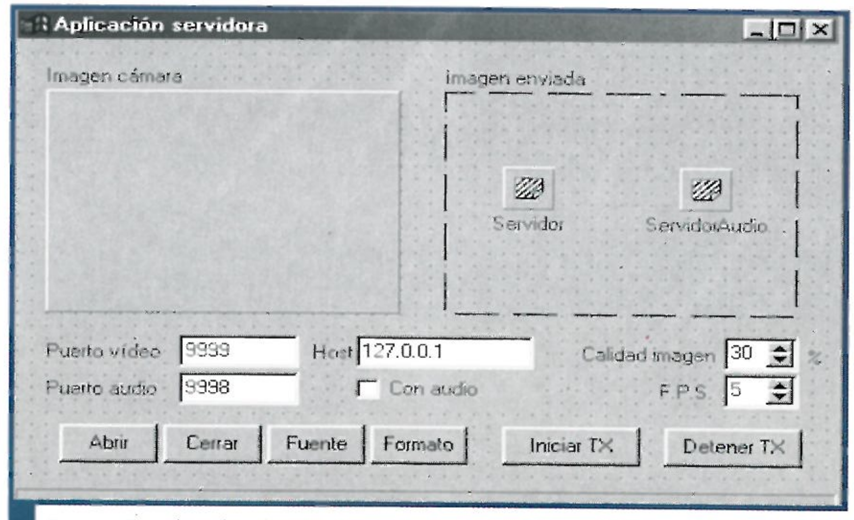


Figura 3. Ejemplo de la aplicación de videoconferencia (visto desde el servidor).



SUSCRÍBETE

A SÓLO PROGRAMADORES

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

- NOTICIAS
- MULTIMEDIA
- TEORÍA
- INTERNET
- DESARROLLO DE APLICACIONES

QUE NO TE LÍEN CON EL <CÓDIGO>

- COMUNICACIONES
- HERRAMIENTAS
- PROGRAMACIÓN
- ÚLTIMAS TENDENCIAS
- Y MUCHO MÁS...





BOLETÍN DE SUSCRIPCIÓN

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L. (Revista Sólo Programadores). C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revista SOLO PROGRAMADORES y beneficiarme de las condiciones de estas magníficas promociónes:

❏ SUSCRIPCION ANUAL
12 NÚMEROS + 12 CD-ROMs
AL PRECIO DE
9.500 ptas. / 57,09 €

❏ SUSCRIPCION ANUAL ESPECIAL ESTUDIANTES
12 NÚMEROS + 12 CD-ROMs
por sólo 7.600 ptas. / 45,67 €

FORMAS DE PAGO:

❏ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❏ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❏ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❏ Domiciliación bancaria
❏ Contra reembolso

Soy antiguo suscriptor
❏ Sí ❏ No

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:
❏ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❏ Transferencia al Banco Popular Español. C/ Valdecanillas, 41. Nº c/c: 0075/1040/43/ 0600047439
❏ Eurocheque conformado con un banco español a nombre de REVISTAS PROFESIONALES, S.L

NOMBRE Y APELLIDOS:...
EDAD:............... PROFESIÓN: ....
TFNO: .... DOMICILIO: ....
CIUDAD:....................................... C.P: ....PROVINCIA:..............................

Datos de domiciliación:
Banco..
Domicilio ....
Nº de Cuenta...
Titular..
Fecha ....
FIRMA

Promoción válida hasta agotar existencias

Desarrollo de aplicaciones con videoconferencia (y V)

Constantino Sánchez Ballesteros (constantino@nexo.es)

En esta última entrega intentaremos comprender cómo trabajan las interfaces *COM* de *Netmeeting*. Así lograremos que los ejemplos incluidos en el SDK sean más comprensibles a la hora de estudiar los desarrollos.



Imaginemos que necesitamos insertar capacidades de conferencia y colaboración en nuestra aplicación. Decidimos utilizar *Microsoft NetMeeting 2.1*. Buena elección, este *software* tiene amplias posibilidades, utiliza interfaces *COM* para que nuestra aplicación sea extensible con la incorporación de componentes dinámicos y no hay que pagar ningún tipo de royaltie.

Después de descargar el *SDK* de *Netmeeting* que encontraremos en la dirección **http://www.microsoft.com/netmeeting/sdk/** ya estamos preparados para empezar a trabajar. Con la referencia de objetos *COM* en la mano y una amplia variedad de ejemplos, muy funcionales, procedemos a crear nuestras propias aplicaciones de videoconferencia.

Necesitamos conocer COM para crear nuestras aplicaciones

Muy pronto descubrimos que los ejemplos, aunque sorprendentes, no cubren exactamente nuestras necesidades (los ejemplos nunca lo hacen). El material de referencia, aunque muy completo en la descripción de interfaces, no explica claramente las relaciones entre los objetos *COM* de *NetMeeting*. Incluso realizando de forma exacta las llamadas a los métodos de las interfaces vemos que no crean los efectos deseados o simplemente fallan por alguna extraña razón. ¿A qué se debe esto? Eso es exactamente lo que vamos a descubrir. Seguiremos las secuencias de eventos de *Netmeeting* y descubriremos algunas idiosincrasias de trabajo en los objetos *COM*.

Es necesario que el lector tenga unos conocimientos de *COM* para poder crear sus propias aplicaciones de conferencia. Si no es así, o el lector ataca el problema de videoconferencia desde *Visual Basic* (este artículo está basado en *C++*), o puede considerar el uso del control *ActiveX* de videoconferencia incluido con *Netmeeting*, que aunque no ofrezca toda la potencia de la programación bajo *C++*, seguramente colmará las exigencias de muchos.



INTRODUCCIÓN A LOS OBJETOS DE NETMEETING

Para acceder a las interfaces y llamar a los métodos que los objetos de *Netmeeting* exponen, necesitamos incluir las declaraciones de los métodos de las interfaces en nuestro proyecto. La forma más fácil de hacerlo consiste en incluir la declaración de archivo **imsconf2.idl** directamente en nuestro proyecto, al igual que hacen todos los ejemplos incluidos en el *SDK*.

Mediante la inclusión de este archivo todos los prototipos de métodos serán generados por el compilador. Una vez que este archivo está en el directorio de proyecto, necesitaremos agregar un paso "*Custom Build*" en el entorno del compilador para que el archivo **imsconf2.idl** genere las apropiadas declaraciones de funciones y declaraciones *GUID* para nuestra propia aplicación.

El objeto Conferencia es el más importante de los incluidos en Netmeeting

Abrimos nuestro proyecto y seleccionamos la opción "*Custom Build*" en la ventana de proyectos. Debería aparecernos una línea que contenga lo siguiente:

```
midl /client none /server none
     /ms_ext /c_ext /env win32 /Os
     /dlldata
$(TargetDir)\dlldata.rpc /proxy $
     (TargetDir)\$(InputName).rpc
     /header $(InputName).h /iid
     $(InputName).c
$(InputPath)
```

Este comando forzará al compilador para que se generen los archivos **imsconf2.c** e **imsconf2.h**. Los dos archivos generados contendrán todos los *GUID's* y declaraciones de funciones de *Netmeeting*.

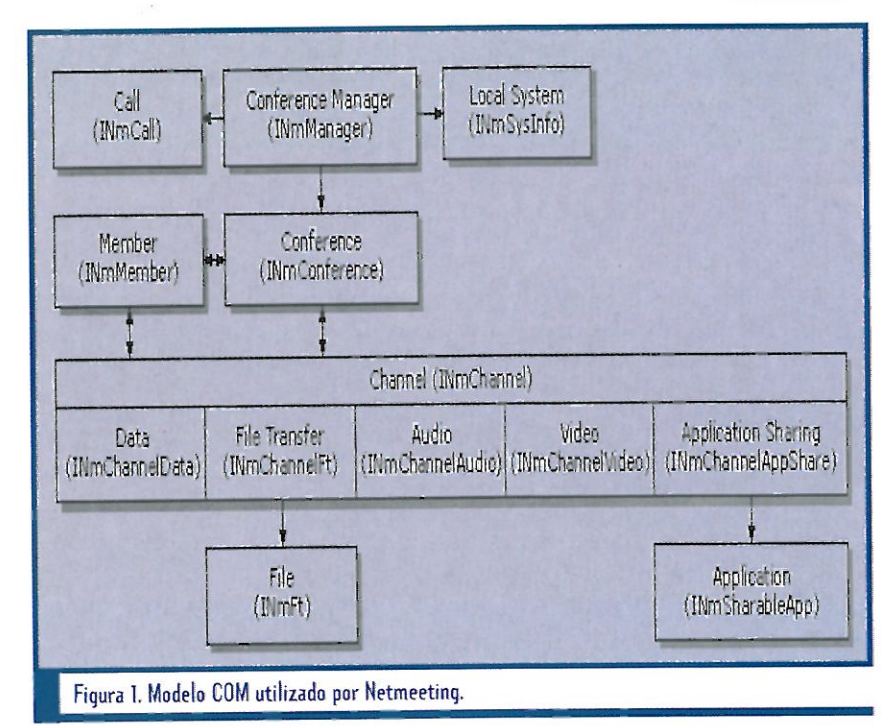


Figura 1. Modelo COM utilizado por Netmeeting.

Puesto que estos archivos no deberían cambiar, el siguiente paso será agregar las siguientes líneas a nuestro archivo **stdafx.h**:

```
#include "imsconf2.c"
// contiene las declaraciones CLSID
#include "imsconf2.h"
// contiene las declaraciones de
     //interfaz
```

Ahora ya podemos decir que nuestro proyecto tiene acceso total a las interfaces *COM* de *Netmeeting*.

Antes de que comencemos con los objetos *COM* de *Netmeeting* necesitamos discutir unas clases base que se utilizan en los ejemplos de la *API*. La primera clase que vamos a ver es **RefCount**. Esta clase, como su nombre implica, gestiona el contador de referencias realizado sobre diversas clases y permite que éstas se autodestruyan cuando su referencia sea **0**. Cada clase que exponga una interfaz *COM* de *NetMeeting 2.1* derivará, por lo menos, de esta clase.

La declaración e implementación de la clase **RefCount** es:

```
class RefCount {
private:
   LONG m_cRef;

public:
  RefCount();
  // el destructor virtual difiere
  // al destructor de la clase
  //derivada.
  virtual ~RefCount();

  // Métodos IUnknown
  ULONG STDMETHODCALLTYPE AddRef();
  ULONG STDMETHODCALLTYPE Release();
};
```

Esta clase es implementada de la siguiente forma:

```
RefCount::RefCount() {
  m_cRef = 1;
}
```

SÓLO PROGRAMADORES

```
RefCount::~RefCount() {
}


ULONG STDMETHODCALLTYPE
      RefCount::AddRef() {
  ASSERT(m_cRef >= 0);
   ::InterlockedIncrement(&m_cRef);
  return (ULONG) m_cRef;
}

ULONG STDMETHODCALLTYPE
  RefCount::Release() {
  if  (::InterlockedDecrement(
              &m_cRef) == 0)
    {
     delete this;
     return 0;
    }

  ASSERT(m_cRef > 0);
  // Si no utilizamos MFC debemos
  // eliminar esta línea.
  return (ULONG) m_cRef;
}
```

La segunda clase que utilizaremos es la clase base para nuestras notificaciones. Esta clase gestiona la conexión y desconexión (*Advise()/Unadvise()*) de los puntos de las interfaces. Cada clase que envuelva una interfaz de notificación derivará de ella.

La declaración de la clase *CNotify* es lo que *se* se lista a continuación:

```
class CNotify {
private:
  DWORD               m_dwCookie;
  IConnectionPoint* m_pcnp;

public:
  CNotify(void);
  ~CNotify();

  HRESULT Connect(IUnknown* pUnk,
                        REFIID    riid,
  IUnknown* pUnkN);
  HRESULT Disconnect();
};
```

Y el código de implementación es como sigue:

```
CNotify::CNotify() {
  m_pcnp = NULL;
  m_dwCookie = 0;
}
CNotify::~CNotify() {
  Disconnect();

  // Nos aseguramos de que estamos
  //desconectados.
}


HRESULT CNotify::Connect (
  IUnknown* pUnk,
  REFIID     riid,
  IUnknown* pUnkN) {

   HRESULT hr;
   ASSERT(m_dwCookie == 0);
   // Nos aseguramos de que lo
   //hacemos sólo una vez

   // Obtenemos el contenedor del
   // punto de conexión
   IConnectionPointContainer*
        pcnpcnt;
   hr = pUnk->QueryInterface(
     IID_IConnectionPointContainer,
     reinterpret_cast<void**>(
        &pcnpcnt));

   if (SUCCEEDED(hr)) {
   // Buscamos el punto de conexión
   // apropiado
  hr = pcnpcnt->FindConnectionPoint
        (riid, &m_pcnp);

  if (SUCCEEDED(hr)) {
   ASSERT(m_pcnp != NULL);
   // Conectamos el objeto
   hr = m_pcnp->Advise(
    reinterpret_cast<IUnknown*>(
    pUnkN), &m_dwCookie);
   }
  pcnpcnt->Release();
 }

 if (FAILED(hr)) {
      m_dwCookie = 0;
 }

return hr;
}

HRESULT CNotify::Disconnect() {
 if (m_dwCookie != 0) {
    // Desconectamos el objeto
    m_pcnp->Unadvise(m_dwCookie);
    m_dwCookie = 0;
    m_pcnp->Release();
    m_pcnp = NULL;
  }

 return S_OK;
}
```

RefCount y CNotify son clases tipo para el resto de las que se basen en Netmeeting

Ahora ya estamos preparados para adentrarnos en los objetos *COM* de *Netmeeting*, comenzando con el principal: El *Manager* de Conferencia concido como *Conference Manager*.

CONFERENCE MANAGER

Es el objeto central de *NetMeeting*. Los demás pueden ser originados o accedidos a través de este objeto. El objeto *Conference Manager* expone una interfaz denominada **INmManager**. Para nuestros ejemplos, crearemos nuestra propia clase que albergue el puntero a la interfaz *INmManager*. Esta clase *C++* será llamada *CconferenceManager* que actuará como el cliente del objeto interno *Conference Manager*. Este objeto llama a las aplicaciones cliente a través de un punto de conexión: la interfaz *INmManagerNotify*.

NOTA: *El objeto Conference Manager notifica cuándo una conferencia o llamada se ha iniciado en el Conference Manager mediante la llamada a los métodos de la interfaz INmManagerNotify que hayamos creado nosotros.*



De manera acorde, nuestra clase *CConferenceManager* tendrá una clase interna que implemente la interfaz *INmManagerNotify* para que el *Conference Manager* pueda llamarla. Esta clase interna de *CConferenceManager* la llamaremos *CConferenceManagerNotify*. Será responsable de la implementación de la interfaz *INmManagerNotify*. El diseño de *CConferenceManager* y su clase interna *CConferenceManagerNotify* se encargarán de manejar todas las conexiones. La declaración de nuestra clase *CConferenceManager* se lista a continuación:

```
class CconferenceManager {
private:
  INmManager* m_pINmManager;
private:
   class CConferenceManagerNotify :
     public RefCount,

     public CNotify,

     public InmManagerNotify {
  private:
   CConferenceManager*
     m_pOwnerManager;
  private:
   CConferenceManagerNotify(
     CConferenceManager*
         pOwnerManager);
   ~CConferenceManagerNotify();

   // INmManagerNotify - Debemos
   // implementar estos métodos
   HRESULT STDMETHODCALLTYPE
      NmUI(CONFN confn);
   HRESULT STDMETHODCALLTYPE
   ConferenceCreated(
     INmConference *pConference);
   HRESULT STDMETHODCALLTYPE
     CallCreated(INmCall *pCall);

   // CNotify
   HRESULT Connect (
        IUnknown *pUnk);

   // Métodos IUnknown, AddRef() y
   // Release() son implementados
   // en la clase base RefCont
   HRESULT STDMETHODCALLTYPE
     QueryInterface(
       REFIID riid, PVOID *ppvObj);

  friend class CConferenceManager;

   };

   CConferenceManagerNotify*
     m_pConfManagerNotify;

public:
  CConferenceManager();
  ~CConferenceManager();

  HRESULT Initialize();
  HRESULT CreateConference(
    BSTR bstrConfName,
    BSTR bstrPassword);
  HRESULT CreateCall(
    BSTR bstrHostAddress,
    INmConference* pINmConference);
  HRESULT AttachConference(
    INmConference* pINmConference);
  HRESULT AttachCall(
    INmCall* pINmCall);
};
```

Como ya se mencionó, *CConferenceManagerNotify*, que implementa a *INmManagerNotify*, es una clase privada en *CConferenceManager*. Además de derivar de **INmManagerNotify**, también lo hace de nuestra clase *RefCount* para gestionar el contador de referencia (después de todo, esto es una implementación de objeto *COM*!), de *CNotify* para gestionar la lógica de conexión, y finalmente, de la interfaz *INmManagerNotify*.

Cuando los datos llegan a través de la interfaz *INmManagerNotify* se entregarán a la clase *CConferenceManagerNotify*. Para mover los datos a nuestra clase *CConferenceManager* necesitaremos tener un puntero a la clase *CConferenceManager* dentro de nuestra clase interna. Este puntero del poseedor del objeto *CConferenceManager* será pasado en el constructor de *CConferenceManagerNotify*.

Echemos un vistazo a los miembros y métodos de *CConferenceManagerNotify*. El primer método que examinaremos será *CConferenceManager:: CConferenceManagerNotify:: Connect*, el cual

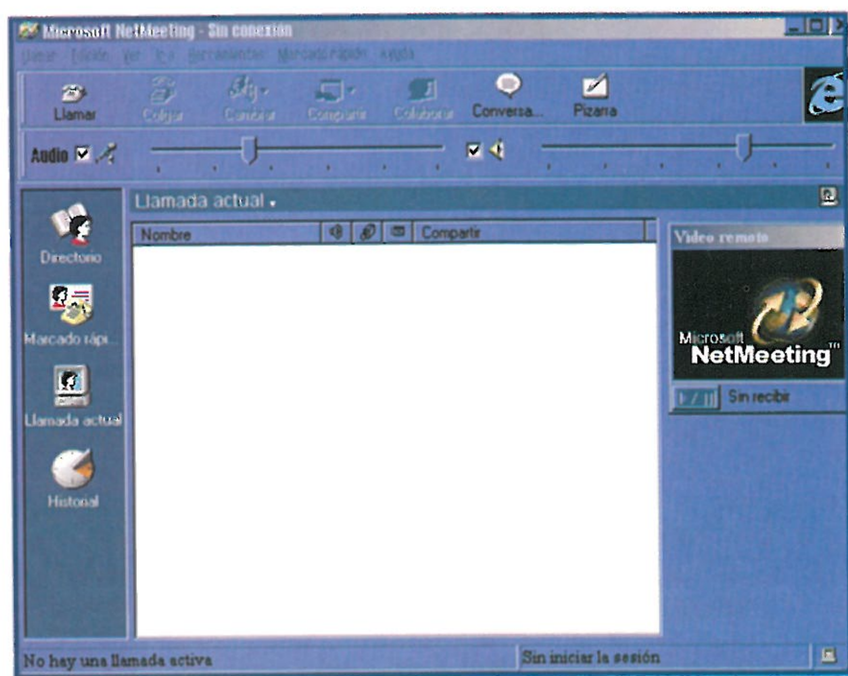


Figura 2. Netmeeting en espera de una llamada. En la ventana derecha podremos visualizar vídeo en tiempo real de otras personas en conferencia.

no hace otra cosa que llamar a su clase base *Connect*. La clase base obtiene el *IUnknown* de *INmManagerNotify*.

El código se muestra a continuación:

```
HRESULT CConferenceManager::
     CConferenceManagerNotify::
     Connect (

 IUnknown* pUnk) {
   return CNotify::Connect(pUnk,
      IID_INmManagerNotify,
      reinterpret_cast<IUnknown*>(
              this));
}
```

El segundo método que vamos a implementar es el estándar *QueryInterface* de **IUnknown:**

```
HRESULT STDMETHODCALLTYPE
     CConferenceManager::
     CConferenceManagerNotify::
     QueryInterface(
 REFIID riid,
 PVOID* ppvObject) {

 HRESULT hr = S_OK;
if (riid == IID_IUnknown) {
 *ppvObject = (IUnknown*)this
 }
else
if (riid == IID_INmManagerNotify){
 *ppvObject =
     (INmManagerNotify*)this;
}
else {
 hr = E_NOINTERFACE;
 *ppvObject = NULL;
}

if (hr == S_OK) {
  AddRef();
}

return hr;}
```

NOTA*: No implementamos AddRef() y Release() porque son gestionados por la clase base RefCount.*

Ahora estamos preparados para comenzar a hablar sobre la implementación de la clase *CConferenceManager*. Esta implementación requerirá instanciar *INmManager* y la clase *CConferenceManager Notify* que implementa a *INManagerNotify*:

Figura 3. Detalle del control ActiveX de conferencia para Visual Basic.

```
CConferenceManager::
    CConferenceManager() {

  m_pINmManager = NULL;
  m_pConfManagerNotify = NULL;
}
```

El destructor gestiona todo el código de limpieza y asegura que el método *UnAdvise* de *INmManagerNotify* sea llamado a través del método *Disconnect* de *Cconference-ManagerNotify*.

```
CConferenceManager::
  ~CConferenceManager() {

// Liberamos nuestro objeto
// de notificación
if (m_pConfManagerNotify
!=NULL) {
m_pConfManagerNotify->Disconnect();

// Puesto que esta clase deriva
// de RefCount, se autodestruirá
// después de Release()
 m_pConfManagerNotify->Release();

 m_pConfManagerNotify = NULL;
}

 // liberamos el objeto COM de
 // Netmeeting
 if (m_pINmManager != NULL) {
  m_pINmManager->Release();
  m_pINmManager = NULL;
 }
}
```

Todas las interfaces de Netmeeting se basan en el modelo COM

El método *CConferenceManager::Initialize()* necesita ser explicado antes de mostrar la implementación. Ocasionalmente, una aplicación que no trabaje adecuadamente puede causar que los objetos *COM* de *Netmeeting* se queden en un estado de limbo. Si *Netmeeting* se encuentra en ese estado cuando comienza nuestra aplicación, *InmManager* se creará y nuestra propia interfaz (*INmManagerNotify*) se conectará propiamente a partir del objeto de conferencia, sin embargo *INmManager::Initialize* fallará. La mejor solución que se puede aplicar es cerrar todo y repetir los pasos de comienzo de una sesión *Netmeeting*. Ahora vamos a crear un bucle que asegure que nuestras aplicaciones *Netmeeting* no fallen cuando una aplicación trabaje incorrectamente. El método *CConferenceManager::Initialize()* se lista a continuación:

```
const UINT MAX_ATTEMPTS = 2; //número
     de intentos

HRESULT CConferenceManager::
     Initialize() {
```



```
HRESULT hr;
LPCLASSFACTORY pcf;

BOOL bSuccess = FALSE;
for (int i = 0;
      i < MAX_ATTEMPTS; i++) {
// Notificamos al sistema que
// queremos utilizar los servicios
// de conferencia mediante la
// creación de un objeto de
// conferencia
hr = ::CoGetClassObject(CLSID_NmMana-
    ger,
 CLSCTX_INPROC, NULL,
 IID_IClassFactory,
 (void**)&pcf);

if (SUCCEEDED(hr)) {
// Obtenemos el objeto Manager de
// Conferencia (conference manager)
 hr = pcf->CreateInstance(NULL,
       IID_INmManager,
       (void**)&m_pINmManager);
 if (SUCCEEDED(hr)) {

 // Conectamos al objeto
 // conference manager
 m_pConfManagerNotify = new
   CConferenceManagerNotify(this);
 if (m_pConfManagerNotify!=NULL) {

 // llamamos al método Connect de
 // CConferenceNotify
 hr = m_pConfManagerNotify->
      Connect(m_pINmManager);
 if (SUCCEEDED(hr)) {
  ULONG uchCaps = NMCH_ALL;
  ULONG uOptions= NM_INIT_CONTROL;
  //Lo inicializamos!!
  hr = m_pINmManager->Initialize(
        &uOptions, &uchCaps);
  bSuccess = SUCCEEDED(hr);
  if (FAILED(hr)) {
m_pConfManagerNotify->Disconnect();
m_pConfManagerNotify->Release();
m_pConfManagerNotify = NULL;
        }
       }
      }
     }
   pcf->Release();
   }

  if (bSuccess)
    break;
  }

 return hr;
}
```

NOTA: *Es muy importante saber que m_pConfManagerNotify->Connect(...) es llamado antes que m_pINmManager->Initialize().*

INmManager::Initialize() merece que también le prestemos un poco de atención. Se pueden apreciar los parámetros que enviamos cuando el método fue llamado:

```
ULONG uchCaps = NMCH_ALL;
ULONG uOptions= NM_INIT_CONTROL;
hr =
m_pINmManager->Initialize(
   &uOptions, &uchCaps);
```

El primer parámetro especifica los canales en los que estamos interesados. *NMCH_ALL* significa que queremos utilizar todos los canales (cinco) que soporta *Netmeeting* (vídeo, audio, transferencia de archivos, colaboración y datos). Si por alguna razón sólo quisiéramos utilizar una serie de canales específicos, deberíamos establecer una combinación de *flags* individuales. Por ejemplo, si queremos utilizar canales de vídeo y datos solamente, utilizaríamos la siguiente expresión:

```
ULONG uchCaps =
  NMCH_VIDEO | NMCH_DATA;
```

El segundo parámetro es utilizado para decirle a *Netmeeting* cómo iniciar sus valores internos. Existen tres elecciones:

1. *NM_INIT_NORMAL* provoca que la consola de *Netmeeting* sea lanzada.
2. *NM_INIT_CONTROL* suprime la consola de *NetMeeting*.
3. *NM_INIT_NO_LAUNCH* provoca que *NetMeeting* no comience a menos que lo haya hecho ya.

Dado que queremos tener un control completo sobre la interfaz de usuario de *Netmeeting*, utilizamos en el ejemplo el *flag NM_INIT_CONTROL*.

Ahora que hemos visto cómo trabaja *CConferenceManager*, es el momento de mostrar la implementación de *CConferenceManagerNotify*.

NOTA: *El objeto interno* Conference Manager *notifica cuándo una conferencia o llamada es iniciada en el Conference Manager llamando a los métodos de la interfaz INmManagerNotify que nosotros mismos hemos implementado.*

Actualmente existen tres tipos de eventos que podrían ser reportados a través de la interfaz **INmManagerNotify**:

1. Se crea un nuevo objeto de conferencia. Este evento podría ser controlado de alguna de las siguientes formas:

- Iniciamos una conferencia llamando al método *INmManager::CreateConference()*.
- Una máquina remota inicia una conferencia colocando una llamada en nuestra máquina.

Para cada una de estas opciones, el método *INmManagerNotify::ConferenceCreated(INmConference* pINmConference)* es llamado en nuestras notificaciones. Llegados a este punto podemos tener acceso al objeto *INmConference* a través del parámetro de esta llamada:

```
HRESULT STDMETHODCALLTYPE CConference-
    Manager::
  CConferenceManagerNotify::
    ConferenceCreated (
     INmConference* pIConference) {

 if (m_pOwnerManager != NULL)
  return
m_pOwnerManager->AttachConference(
```

```
    pIConference);
  else
   return E_FAIL;
}
```

NOTA: *se presume que la clase CConferenceManager implementa el método AttachConference() que crea una instancia de una clase C++ para los objetos de conferencia de Netmeeting.*

2. Se crea un objeto de llamada. Este evento se puede controlar de dos formas:

- Iniciamos una llamada con el método *INmManager::CreateCall()*.
- Una máquina remota inicia una llamada colocando ésta última en nuestra máquina.

En este caso es el método *INmManagerNotify::CallCreated(INmCall* pINmCall)* el llamado. Al igual que antes, en este punto tenemos acceso al objeto de llamada a través del parámetro de la misma:

```
HRESULT STDMETHODCALLTYPE CConference-
     Manager::
  CConferenceManagerNotify::
    CallCreated(INmCall* pICall) {

if (m_pOwnerManager != NULL)
 return
 m_pOwnerManager->AttachCall(
       pICall);
else
 return E_FAIL;
}
```

3. *Conference Manager* puede necesitar una actualización de su cambio de estado interno. En este caso, llamaríamos al método *INmManagerNotify::NmUI(CONFN confn)*.

Una vez que tenemos un plan para gestionar los tres tipos de eventos que pueden ser devueltos, especialmente la notificación *ConferenceCreated*, podríamos preguntarnos cómo comenzar la conferencia. El código de comienzo de una conferencia es similar al siguiente:

Figura 4. Clases disponibles del control de conferencia para Visual Basic.

```
HRESULT CConferenceManager::
  CreateConference(
   BSTR bstrConfName,
   BSTR bstrPassword) {

// Primero nos aseguramos de que
// tenemos un objeto Conference
// Manager válido
if (m_pINmManager == NULL)
  return E_FAIL;

 INmConference* pINmConference =
   NULL;

HRESULT hr =
 m_pINmManager->CreateConference(
    &pINmConference, bstrConfName,
bstrPassword, NMCH_ALL);

return hr;
}
```

El método *CreateConference* llama al método *INmManager::CreateConference(...)*. Como resultado *NetMeeting* crea un objeto de conferencia interno y devuelve su puntero a la interfaz *INmConference* en el primer parámetro. El método *INmManager::CreateConference()* también toma el nombre de la conferencia, la contraseña y un *flag* que indica los tipos de canales utilizados para la conferencia. Las llamadas de *NetMeeting*, al igual que las que podemos ver en *Internet Explorer*, pueden ser instanciadas de forma similar (pero al igual que las conferencias de *NetMeeting* no están protegidas por contraseña):

```
HRESULT CConferenceManager::
   CreateCall(
   BSTR bstrHostAddress,
   INmConference* pINmConference) {
if (m_pINmManager == NULL ||
     pIConference  == NULL)
 return E_FAIL;

 INmCall* pICall = NULL;
 HRESULT hr =
 m_pINmManager->CreateCall(
    &pINmCall, NM_CALL_DEFAULT,
    NM_ADDR_IP, bstrHostAddress,
    pINmConference);
 return hr;
 }
```

Cuando llamamos al método *INmManager::CreateCall()*, *NetMeeting* crea un objeto de llamada interno y retorna su puntero a la interfaz *INmCall* en el primer parámetro. El lector podría preguntarse por qué pasamos una interfaz para que sea inicializada y no hacemos nada con ella. La razón se refiere a la consistencia. Nosotros permitimos a nuestra interfaz que gestione esos objetos en los métodos *ConferenceCreated* y *CallCreated*. Ahora que hemos visto los aspectos básicos de *NetMeeting*, estamos en disposición de revisar conferencias, canales y canales de datos.

CONFERENCIAS, CANALES Y CANALES DE DATOS

Siempre que se instancia un objeto de conferencia, *NetMeeting* crea automáticamente los canales de audio, vídeo, transferencia de archivos y canales de colaboración y nos notifica su creación a través de *INmConferenceNotify*. Estos canales sólo existen dentro del contexto del objeto de conferencia. Cuando la llamada



a *AttachConference* se recibe, nosotros ajustamos el objeto entrante *INmConference* con nuestra propia clase *C++*. Ésta aplicará exactamente la misma aproximación utilizada para el objeto *Conference Manager*. Esta clase se llamará *CConference*. Actuará como cliente al objeto interno de conferencia manteniendo una interfaz *INmConferenceNotify*.

NOTA: *Dado que se nos notifica a través de la interfaz INmConferenceNotify, es nuestra responsabilidad implementarla y gestionar sus métodos.*

Por ejemplo, cuando se crea un canal de audio, se nos notifica mediante *INmConferenceNotify::ChannelChanged(...)*. Cuando un *INmChannel* llega a nosotros a través de este método, es nuestra responsabilidad crear nuestra propia clase o miembro de clase para mantener el puntero a la interfaz. Una implementación de *ChannelChanged* podría ser como sigue:

```
HRESULT STDMETHODCALLTYPE
CConference::
CConferenceNotify::
ChannelChanged(
 NM_CHANNEL_NOTIFY uNotify,
 INmChannel* pIChannel) {

// Primero, validamos todos
// los punteros
if (m_pOwnerConference == NULL ||
    m_pOwnerConference->
    m_pINmConference == NULL ||
    pIChannel == NULL)
 return E_FAIL;

HRESULT hr = S_OK;
switch (uNotify) {
 case NM_CHANNEL_ADDED:
  hr =
 m_pOwnerConference->
    AttachChannel(pIChannel);
  break;
 case NM_CHANNEL_REMOVED:
  hr =
 m_pOwnerConference->DetachChannel(
     pIChannel);
  break;

 case NM_CHANNEL_UPDATED:
  hr =
 m_pOwnerConference->UpdateChannel(
     pIChannel);
  break;
}

 return hr;
}
```

El código de ejemplo anterior provee tres métodos con nuestra clase *CConference*:

1. *AttachChannel (INmChannel* pIChannel)* gestionará la creación de objetos para los diversos canales. A continuación se muestra cómo implementar *AttachChannel*:

```
HRESULT CConference::
AttachChannel(
  INmChannel* pIChannel) {
// Determina el tipo de canal
ULONG uType;

HRESULT hr =
pIChannel->GetNmch(&uType);
if (FAILED(hr)){
 return hr;
}

switch (uType){
 case NMCH_VIDEO:
 // Hacemos algo para el
 // canal de vídeo
 //...
 break;

 case NMCH_AUDIO:
 // Hacemos algo para el
 // canal de audio
 //...
 break;

 case NMCH_DATA:
 // Hacemos algo para el
 // canal de datos
 //...
 break;

 case NMCH_FT:
 // Hacemos algo para el canal
 // de transferencia de archivos
 //...
 break;

 case NMCH_SHARE:
 // Hacemos algo para el canal
 // de colaboración
 //...
 break;
}

 return S_OK;
}
```

2. *DetachChannel (INmChannel* pIChannel)* se le llamará cuando se destruya un canal.

3. *UpdateChannel (INmChannel* pIChannel)* notificará al cliente del objeto de conferencia de *Netmeeting* las actualizaciones que están ocurriendo en los canales específicos.

El canal de datos (*Data Channel*), a diferencia del resto, se crea de forma automática. Puesto que podemos crear un número ilimitado de canales de datos y cada uno de ellos se representa por un único *GUID*. Por ejemplo, si dos aplicaciones están participando en una conferencia y ambas crean un canal de datos con el mismo *GUID*, los dos canales de datos se comunicarán el uno con el otro. Siguiendo esta metodología, si una aplicación *A* crea un canal de datos con un *GUID ABC* y una aplicación *B* crea una con el *GUID XYZ*, los dos canales de datos no hablarán el uno con el otro. Esto refleja perfectamente el uso de *GUID's*.

Podemos utilizar el canal de datos para gestionar cualquier cosa, desde comandos hasta datos. La clase *CConference* que se crea en respuesta al *NetMeeting:INmConference* que está siendo creado, es una buena clase para implementar un canal de datos, ya que típicamente tendríamos una conferencia establecida si requiriéramos un canal de datos.





XML (IV). El lenguaje XSL (II)

Adolfo Aladro García (aaladro@arrakis.es)

El lenguaje XSL proporciona una manera excelente de realizar consultas y buscar información en una fuente de datos XML. Gracias a los patrones XSL es posible presentar la información filtrándola con las normas que deseemos.

INTRODUCCIÓN

En la entrega anterior se introdujo el lenguaje *XSL*. Vimos que se trata de un "puente" mediante el que podemos relacionar los contenidos (o sea, las fuentes de datos *XML*) con la presentación de los mismos (las páginas *HTML* donde normalmente se visualizará la información). Las hojas de estilo *XSL*, donde se entremezclan etiquetas *HTML* y *XSL*, son además en sí mismas documentos *XML*. Es importante tener en cuenta esto ya que las hojas deben cumplir las normas impuestas por el estándar *XML*, tanto en *HTML* como en *XSL*.

Pero el lenguaje *XSL* va mucho allá de la simple presentación de los contenidos. Su sintaxis proporciona un método potente de realizar consultas, búsquedas y filtrado de datos. Se trata de un mecanismo similar al que ofrece el lenguaje *SQL*. Es imprescindible dominar el lenguaje *XSL* para todo aquel desarrollador que desee trabajar con datos *XML*.

SINTAXIS DE LOS PATRONES XSL

Se utilizan para hacer consultas y filtrar la información procedente de una fuente de datos *XML*. Cada patrón define un conjunto de nodos pertenecientes a la jerarquía. Debemos recordar que todo documento *XML* representa un árbol de datos. Un patrón viene a ser una manera de definir selecciones de nodos que forman parte de ese árbol. Para empezar a comprender mejor la sintaxis de los patrones *XSL* vamos a partir de un ejemplo. El Listado 1 muestra el contenido del archivo **biblioteca.xml**. Se trata de una fuente de datos donde se guarda información relativa a una colección de libros. La Figura 1 muestra una representación gráfica del árbol que se deduce del documento anterior. La sintaxis de los patrones *XSL* está fuertemente inspirada en la sintaxis de las direcciones *URL*, pero en vez de reflejar una estructura de directorios y subdirectorios, se trata de representar una serie de nodos y ramas de un árbol de datos. Por ejemplo, el patrón

```
Libro/autor
```

seleccionaría un conjunto formado por todos los elementos *autor* que están contenidos dentro del elemento *libro*. Esta anotación permite realizar consultas al árbol que representa toda la información contenida en un documento *XML*.

EL CONTEXTO

Toda consulta a una fuente de datos *XML* sucede en un determinado contexto, o lo que es lo mismo, se realiza contra un nivel concreto del árbol *XML* de datos. De esta manera, un contexto se define como el nodo a partir del



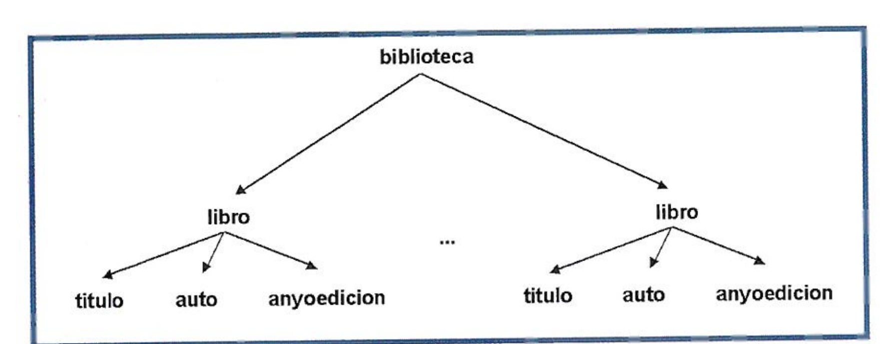


Figura 1. Árbol que representa la fuente de datos **biblioteca.xml**.

Listado1. Fuentes de datos XML que representan una colección de libros (direcciones.xml)

```
<?xml version='1.0'?>
<biblioteca>
  <libro ISBN="1857939727">
    <titulo>Claude Debussy : An Essential Guide to His Life and
Works</titulo>
    <autor>Jonathon Brown</autor>
    <anyoedicion>1997</anyoedicion>
  </libro>
  <libro ISBN="0486229165">
    <titulo>Claude Debussy : His Life and Works</titulo>
    <autor>Leon Vallas</autor>
    <anyoedicion>1973</anyoedicion>
  </libro>
  <libro ISBN="0711917523">
    <titulo>Debussy (The Illustrated Lives of the Great Compo-
sers Series)</titulo>
    <autor>Paul Holmes</autor>
    <anyoedicion>1992</anyoedicion>
  </libro>
  <libro ISBN="0931340411">
    <titulo>Debussy Remembered</titulo>
    <autor>Roger Nichols</autor>
    <anyoedicion>1992</anyoedicion>
  </libro>
  <libro ISBN="1569220662">
    <titulo>Debussy, Very Best for Piano</titulo>
    <autor>John L. Haag</autor>
    <anyoedicion>1998</anyoedicion>
  </libro>
  <libro ISBN="0300044399">
    <titulo>Music of Claude Debussy (Composers of the Twentieth
Century)</titulo>
    <autor>Richard S. Parks</autor>
    <anyoedicion>1990</anyoedicion>
  </libro>
</biblioteca>
```

cual se realiza una consulta mediante un patrón *XSL*. Como es lógico, el resultado de una búsqueda o consulta siempre dependerá del "lugar" dónde estemos buscando. Una consulta basada en un patrón *XSL* puede buscar patrones específicos en un determinado contexto, devolver los resultados y llevar a cabo operaciones adicionales relativas al contexto de los nodos devueltos.

Las hojas de estilo XSL, donde se entremezclan etiquetas HTML y XSL, son además en sí mismas documentos XML

Al mismo tiempo, se utilizarán los propios patrones para establecer el contexto en cada caso. En el artículo anterior vimos el elemento *xls:for-each* del lenguaje *XSL*. El valor del atributo *select* no es más que un patrón *XSL* mediante el cual definimos el contexto de la búsqueda. En concreto, */biblioteca/libro* selecciona todos los libros de la biblioteca.

```
<?xml version='1.0'?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/
    TR/WD-xsl">
<xsl:template match="/">
<HTML>
<BODY>
<xsl:for-each select="/
    biblioteca/libro">
<xsl:value-of
    select="titulo"/><BR/>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

La hoja *XSL* anterior serviría para mostrar todos los títulos

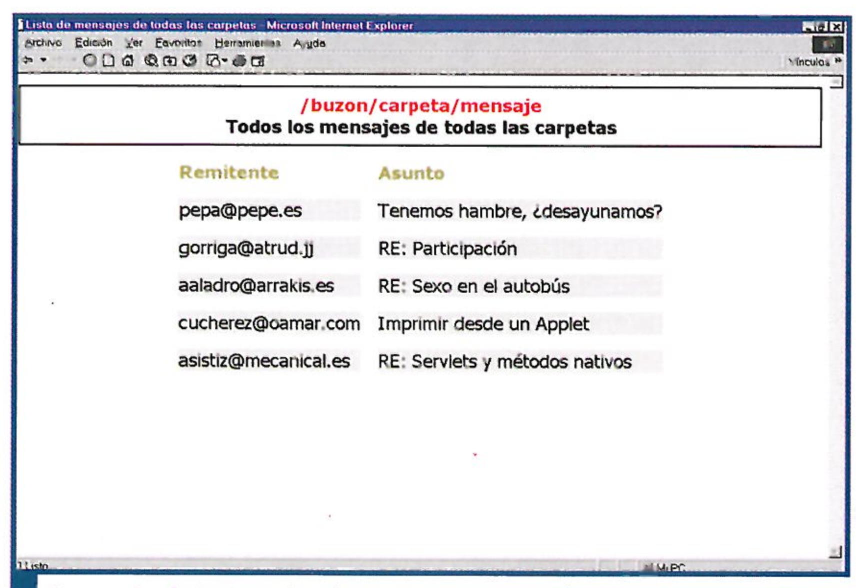


Figura 2. Resultado de visualizar la fuente de datos **buzon.xml** utilizando una hoja de estilo, en concreto **buzon.xsl**, que contiene un patrón simple.

contenidos en la biblioteca. En primer lugar el atributo *match* de la etiqueta *xsl:template* contiene la barra inclinada / para hacer notar que vamos a buscar a partir del nodo raíz del árbol.

COLECCIONES

Dada una lista de nodos es posible referirnos a uno en concreto mediante un patrón *XSL*. Para ello simplemente debemos utilizar corchetes y un valor entero entre **0** y el número de elementos menos uno.

Veamos otro ejemplo. El archivo **buzon.xml** representa un buzón de correo electrónico. Éste está formado por carpetas que contienen mensajes, o más bien deberíamos decir cabeceras de mensajes. El Listado 2, incluido en el CD-ROM, muestra este documento *XML*.

- */buzon/carpeta/mensaje*
  Todos los mensajes de todas las carpetas.
- */buzon/carpeta/mensaje[0]*
  Todos los mensajes que ocupan el primer lugar entre los mensajes que pertenecen a su misma carpeta.
- */buzon/(carpeta/mensaje)[0]*
  El primer mensaje de entre todos los mensajes (Sea cual sea la carpeta a la que pertenecen).
- */buzon/carpeta[1]/mensaje[0]*
  El primer mensaje de la carpeta segunda.

Toda consulta a una fuente de datos XML sucede en un determinado contexto

El Listado 3, también ubicado en el CD-ROM contiene un ejemplo de hoja *XSL* mediante la que podemos visualizar y filtrar los datos procedentes del buzón de correo.

Los ejemplos anteriores de patrones pueden ponerse en el atributos *select* de la etiqueta *xsl:for-each* para verificar que, efectivamente, su comportamiento es el descrito.

FILTROS

Se pueden aplicar filtros a las búsquedas. El resultado es similar a lo que ocurre cuando utilizamos una sentencia *WHERE* en una consulta *SQL*. De hecho podemos observar el código del Listado 3, en el CD, y veremos que en él aparece un filtro. Otros ejemplos de filtros son:

- *buzon/carpeta/mensaje [remitente]*
  Todos los mensajes, sea cual sea la carpeta a la que pertenezcan, que cuenta con un elemento *remitente*.
- */buzon/carpeta[@nombre ='Mensajes personales'] /mensaje*
  Todos los mensajes de la carpeta cuyo atributo *nombre* tenga un valor igual a *Mensajes personales*.

EXPRESIONES BOOLEANAS, DE COMPARACIÓN Y DE ASIGNACIÓN

La Tabla 1 ofrece el conjunto de operadores que podemos utilizar a la hora de construir los patrones *XSL*. Aquellos que se notan con un asterisco (*) pertenecen al grupo de métodos para patrones descritos en la propuesta *XQL*, por lo que no podemos considerar, por el momento, que formen parte del estándar, aunque podemos utilizarlos en el caso de que estemos trabajando con *Internet Explorer 5.0*.

La sintaxis propuesta por *W3C*, en lo que respecta a los operadores, utiliza espacios en blanco y otro tipo de separadores preferentemente, en lugar del carácter de dolar ($). Además, las palabras clave binarias del tipo *xxx* pueden expresarse como *wsxxxws*, donde *ws* se refiere al *token* delimitador, que puede ser espacios en



blanco, comillas simples (') o comillas dobles ("). Los operadores *unarios*, como *not()*, utilizan notacional funcional, lo que quiere decir que se escriben con paréntesis como si de funciones se tratase. Aunque *Microsoft* soporte los dos tipos de sintaxis es conveniente utilizar la propuesta por el *W3C* de cara a mantener la compatibilidad.

El resultado de aplicar un filtro es similar a lo que ocurre cuando utilizamos una sentencia WHERE en una consulta SQL

El orden de precedencia de los operadores de comparación y los operadores booleanos es el que aparece en la Tabla 2.

MÉTODOS

La sintaxis de los patrones *XSL* proporciona métodos. Éstos pueden clasificarse en dos grupos:

A. Métodos de información. Proporcionan información acerca de nodos concretos (tipo de nodo, nombre del nodo, su texto, etc.).

Existen dos tipos de métodos: de información y de colección

B. Métodos de colección. Devuelven una colección con todos los nodos que se ajustan a un determinado tipo (por ejemplo: todos los nodos que son comentarios, todos los nodos que son instrucciones de procesamiento).

Para aplicar un método a un determinado contexto, debemos anexar el carácter de admiración (!) seguido del nombre del método al contexto. Al escribir el nombre hay que tener cuidado porque los métodos distinguen entre mayúsculas y minúsculas.

Tabla 1. Operadores que se pueden utilizar en los patrones XSL.

Operador	Sintáxis alternativa	Atajo	Descripción
and	and	&&	AND lógico
or	or	||	OR lógico
not()	not		Negación
=	eq		Igualdad
	ieq(*)		Igualdad (no distingue entre mayúsculas y minúsculas)
!= (*)	ne		Distinto
	ine		Distinto (no distingue entre mayúsculas y minúsculas)
<*	lt		
ilt*			Menor
			Menor (no distingue entre mayúsculas y minúsculas)
<=*	le		
ile*			Menor o igual
			Menor o igual (no distingue entre mayúsculas y minúsculas)
>*	gt		
igt*			Mayor
			Mayor (no distingue entre mayúsculas y minúsculas)
>=*	ge		
ige*			
all*			
any*			Mayor o igual
			Mayor o igual (no distingue entre mayúsculas y minúsculas)

Tabla 2. Orden de precedencia de los operadores de comparación y de los operadores booleanos

Operador	Descripción
1. ()	Agrupamiento
2. []	Filtros
3. !	Invocación de métodos
4. / //	Operadores de contexto
5. any all	Operadores de asignación
6. = != lt le gt ge eq ne ieq	Comparaciones
7. |	Unión
8. Not()	**NOT**
9. And	*AND*
10. Or	*OR*

A. Métodos de información.

- *date(fecha)*

Convierte valores a fechas. El parámetro *fecha* es una representación de una fecha en el

SÓLO PROGRAMADORES

formato *mm-dd-yy* o en cualquier otro formato *ISO* para fechas soportado por *XML*. El parámetro puede ser un literal o un patrón *XSL*.

- *end()*
Devuelve **true** en el caso de tratarse del último elemento de una colección. Este método es relativo al nodo padre.
- *index()*
Devuelve el índice del nodo con respecto al padre al que pertenece. Este número será un valor que irá desde 0 hasta el número de nodos hijo menos uno. Volviendo al ejemplo proporcionado por la fuente de datos buzon.xml:

```
<xsl:for-each select="/buzon/
    carpeta[index() $lt$ 1]/
    mensaje">
```

La etiqueta anterior seleccionaría todos los mensajes de la carpeta cuyo índice es menor a uno, o sea, la carpeta que ocupa la primera posición en el documento *XML*.

- *nodeName()*
Devuelve un texto con el nombre de la etiqueta del nodo.
- *nodeType()*
Devuelve un número que indica el tipo del nodo.
- *number()*
Convierte un valor dado a un número.
- *value()*
Devuelve una versión con tipo del valor de un elemento. Si el uso de tipos de los elementos no está soportado, este método devuelve la cadena de texto que representa al valor del elemento.

Cuando se están realizando comparaciones el método *value* no se utiliza. Esto es así porque las comparaciones operan por defecto con el valor de los dos elementos que se comparan.

B. Métodos de colección.

- *attribute(nombre)*
Devuelve una colección con todos los nodos de tipo atributo. El parámetro *nombre* es opcional y equivale al nombre del atributo a devolver en la colección. Si se proporciona el parámetro adicional el método devuelve solamente los nodos del tipo atributo que se corresponden con el nombre indicado.
- *cdata ()*
Devuelve una colección con todos los nodos de *CDATA*.

```
//cdata()
```

El método index() devuelve el índice del nodo con respecto al padre al que pertenece

El ejemplo anterior sirve para seleccionar todos los nodos del tipo *CDATA* en el documento.

- *comment()*
Devuelve una colección de nodos del tipo comentario. Si quisiéramos encontrar el segundo comentario que aparece en cualquier punto del documento fuente:

```
//comment()[1]
```

- *element(nombre)*
Devuelve una colección de nodos del tipo elemento. El parámetro *nombre* es opcional y equivale al nombre del elemento a devolver en la colección. Si se proporciona el parámetro adicional el método devuelve solamente los nodos del tipo elemento que se corresponden con el nombre indicado.
- *node()*
Devuelve una colección formada por todos los nodos excepto los del tipo atributo y el nodo raíz del documento.
- *pi(nombre)*
Devuelve una colección de nodos del tipo *PI*. El parámetro *nombre* es opcional y equivale al nombre del elemento a devolver en la colección. Si se proporciona el parámetro

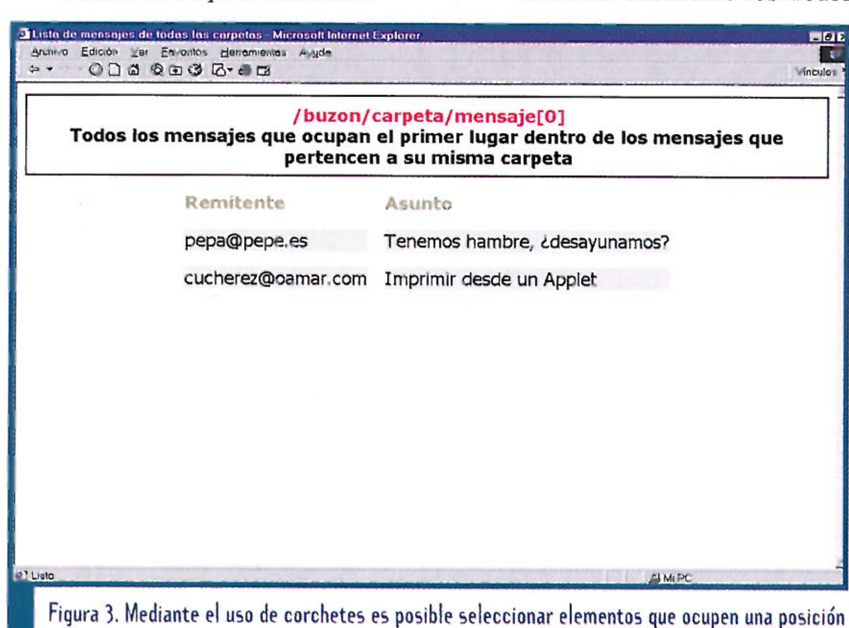


Figura 3. Mediante el uso de corchetes es posible seleccionar elementos que ocupen una posición concreta dentro de una colección.



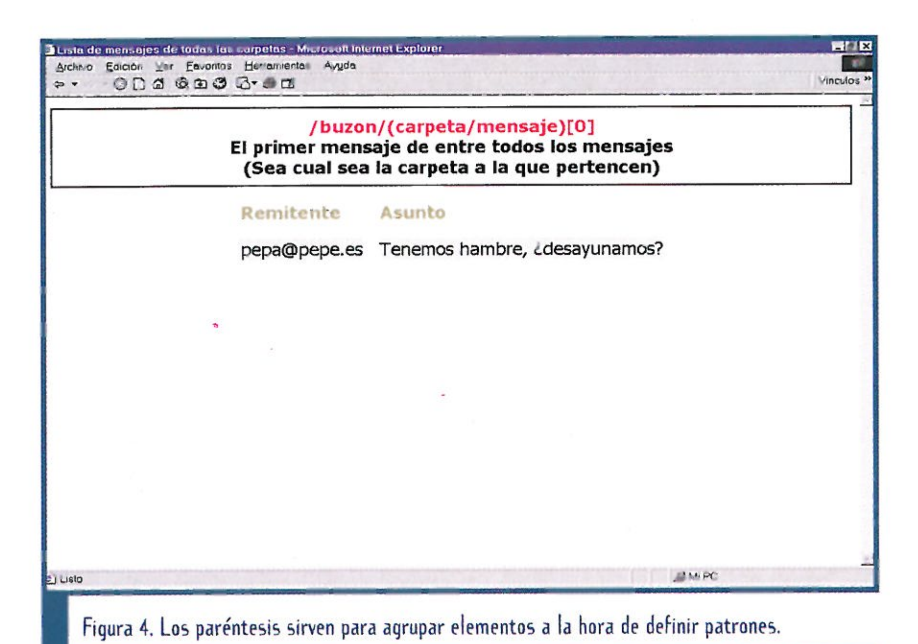


Figura 4. Los paréntesis sirven para agrupar elementos a la hora de definir patrones.

adicional el método devuelve solamente los nodos del tipo *PI* que se corresponden con el nombre indicado.

```
/pi('xml-stylesheet')
```

Las comparaciones operan por defecto con el valor de los dos elementos que se comparan

El ejemplo anterior serviría para seleccionar el nodo correspondiente a la etiqueta:

```
<?xml-stylesheet type="text/xsl"
    href="buzon.xsl"?>
```

- *text()*
  Devuelve una colección de nodos del tipo texto o *CDATA*.
- *textnode()*
  Devuelve una colección de nodos del tipo texto.

XSL DESDE EL DOM

Hasta hora hemos visto que podemos utilizar el lenguaje *XSL* para crear hojas de estilo con las que visualizar directamente los documentos *XML*. Además de esto, es posible utilizar *XSL* desde el *DOM* que proporciona el navegador, es decir, realizando *scripts* de *JavaScript*. La funcionalidad *XSL* proporcionada por el *DOM* se produce a través de cuatro métodos:

- *nodo.selectNodes(patron)*
  Aplica el patrón que recibe como parámetro al contexto definido por *nodo* y devuelve una lista con todos los nodos que cumplen el patrón.
- *nodo.selectSingleNode(patron)*
  Aplica el patrón que recibe como parámetro al contexto definido por *nodo* y devuelve el primer nodo que encuentra que cumple patrón.
- *cadena =nodo.transform Node (hoja_estilo)*
  Procesa el nodo indicado y todos sus hijos utilizando la hoja *XSL* que se le pasa como parámetro. Devuelve una cadena de texto con el resultado de la transformación.
- *nodo.transformNodeToObject (hoja_estilo, objeto_salida)*
  Procesa el nodo indicado y todos sus hijos utilizando la hoja *XSL* que se le pasa como parámetro. Devuelve el resultado de la transformación en el objeto indicado por *objeto_salida*.

El método cdata() devuelve una colección con todos los nodos del tipo CDATA

REALIZAR CONSULTAS CON JAVASCRIPT Y PATRONES XSL

Debemos tener presente en todo momento que una hoja *XSL* es a su vez un documento *XML* y que por lo tanto puede ser manipulada mediante *JavaScript* tal y como vimos en anteriores entregas. En una página *HTML* la incluiremos simplemente utilizando la etiqueta *<XML>*. El ejemplo que vamos a ver a continuación consta de una página *HTML* (**cesta_compra.htm**) y un documento *XML* (**cesta_compra.xml**) en el que se reflejan los productos que un determinado usuario ha "echado" a su cesta de la compra (Esta situación se da frecuentemente en todos los *Web* dedicados al comercio electrónico. De hecho, la tecnología *XML* está empezando a verse en estos entornos como una forma con la que se agilizarán todos los procesos involucrados en las compras a través de *Internet*). Para presentar la cesta de la compra utilizaremos una hoja *XSL* (**cesta_compra.xsl**).

```
<DIV ID="tabla"></DIV>
<DIV ALIGN="RIGHT" ID="sumatotal"
```

```
      STYLE="font-family: Tahoma;
      font-size: 12pt; color: Red;
      font-weight: bold; width:
      80%;"></DIV>
<XML ID="documentoXML"
      SRC="cesta_compra.xml"></XML>
<XML ID="documentoXSL"
      SRC="cesta_compra.xsl"></XML>
```

El cuerpo de la página *HTML* tendrá básicamente el código anterior. En él se definen dos capas de *HTML* mediante la etiqueta *<DIV>*. En la primera, cuyo identificador es *tabla*, aparecerá el código *HTML* resultado de aplicar la hoja de estilo a la fuente de datos. La capa identificada como *sumatotal* contendrá un valor numérico correspondiente al importe total de la compra. Éste será calculado mediante *JavaScript*. Las etiquetas *<XML>* sirven para vincular la fuente de datos así como la hoja de estilo.

```
function iniciar() {
   tabla.innerHTML = documen-
      toXML.transformNode(documen-
      toXSL.XMLDocument);
   gastoTotal();
}
```

Podemos utilizar XSL desde el DOM que proporciona el navegador, es decir, realizando scripts de JavaScript

La función *iniciar* se ejecuta cuando sucede el evento *onload*. El método *transformNode* del objeto que representa al documento *XML* toma como parámetro el objeto que representa a la hoja de estilo y transforma los datos *XML* consecuentemente. El resultado que se obtiene es una cadena de texto. La propiedad *innerHTML* de una capa definida mediante la etiqueta *<DIV>* sirve para actualizar y/o consultar el código *HTML* contenido en dicha capa.

La función *gastoTotal* será la que se encargue de calcular el importe total de la compra. Para ello tendremos que buscar en el árbol XML los nodos etiquetados como *<cantidad>* y *<precio>* y realizar las operaciones pertinentes. Cuando el importe total haya sido calculado se mostrará actualizando la capa que le corresponde. Para ello utilizaremos, tal y como acabamos de hacer ahora, la propiedade *innerHTML* de la capa.

```
Function gastoTotal() {
   Var nodos =
      documentoXML.selectNodes
      ("cesta/articulo");
   Var nodo;
   Var total = 0;

   for(nodo = nodos.nextNode();
      nodo; nodo =
      nodos.nextNode()) {
      total +=
      nodo.childNodes.item(2).
      nodeTypedValue*nodo.
      childNodes.item(3).
      nodeTypedValue;
   }
   sumatotal.innerHTML = "Total =
      " + total;
}
```

Las etiquetas <XML> sirven para vincular la fuente de datos así como la hoja de estilo

En primer lugar utilizamos el método *selectNodes* para obtener una colección de nodos formada por los artículos de la cesta de la compra. Para recorrer esta lista de nodos utilizamos un bucle *for* así como la llamada al método *nextNode*, el cual nos permitirá tener acceso al siguiente nodo de la colección

Listado4. Hoja XSL mediante la cual visualizamos la fuente de datos, correspondiente a la cesta de la compra, dentro de una página HTML

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<xsl:script>
<![CDATA[
  var contador = -1;
  function ActualizarContador() {
    contador++;
    return contador;
  }
]]>
</xsl:script>
<xsl:for-each select="/cesta/articulo">
  <DIV STYLE="font-family : Tahoma; font-size : 12pt; color:
Navy; font-weight: bold;">
    <xsl:eval>ActualizarContador()</xsl:eval>
    <xsl:value-of select="descripcion"/>
  </DIV>
  <DIV STYLE="font-family : Tahoma; font-size : 12pt;">
    C&#243;digo: <xsl:value-of select="codigo"/><BR/>
    Cantidad: <xsl:value-of select="cantidad"/><BR/>
    Precio: <xsl:value-of select="precio"/>
  </DIV>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```



sucesivamente. Cada nodo que obtenemos representa un artículo de la cesta de la compra y por lo tanto debe tener cuatro nodos hijo: *código*, *descripción*, *cantidad* y *precio*. Estos nodos hijos ocuparán respectivamente las posiciones 0, 1, 2 y 3 de la lista de nodos que proporciona *childNodes*.

INCORPORAR JAVASCRIPT A LAS HOJAS XSL

En el caso anterior hemos visto como es posible utilizar el lenguaje *JavaScript* para acceder a la fuente de datos proporcionada por el documento *XML*. Ahora vamos a ver otra forma de utilizar *JavaScript*. Esta vez se trata de incorporar a la hoja de estilo *scripts*. Éstos formarán parte de la propia hoja de estilo y serán muy útiles para dar lugar a estilos más elaborados.

La propiedad innerHTML de una capa sirve para actualizar y/o consultar el código HTML contenido en dicha capa

El Listado 4 muestra la hoja de estilo que hemos utilizado en nuestro ejemplo anterior. Hemos añadido un *script* mediante el que podremos numerar los artículos que constituyen la cesta de la compra. Los *scripts* que se incluyen dentro de hojas XSL deben ser, necesariamente, escapados y para ello se insertan en una sección del tipo *CDATA*. La etiqueta *<xsl:eval>* sirve para poder evaluar el resultado de un expresión por lo que representará la manera en la que invocaremos a nuestras funciones definidas en *JavaScript*.

Los *scripts*, tanto en la propia página *HTML* como en la hoja de estilo *XSL*, son verdaderamente útiles y permiten realizar operaciones tan sofisticadas y complejas como queramos con los datos *XML* y la forma en que éstos se presentan.

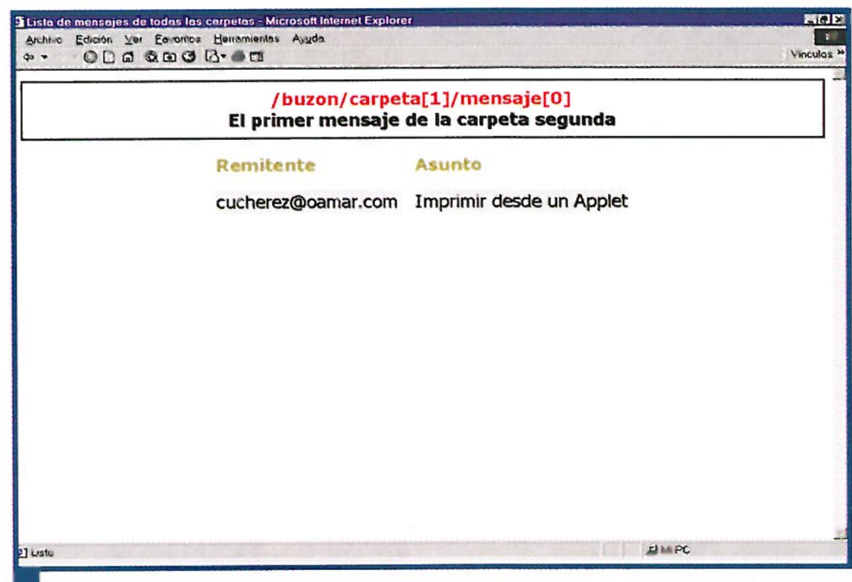


Figura 5. La utilización de patrones nos permite realizar cualquier tipo de consulta contra fuente de datos XML.

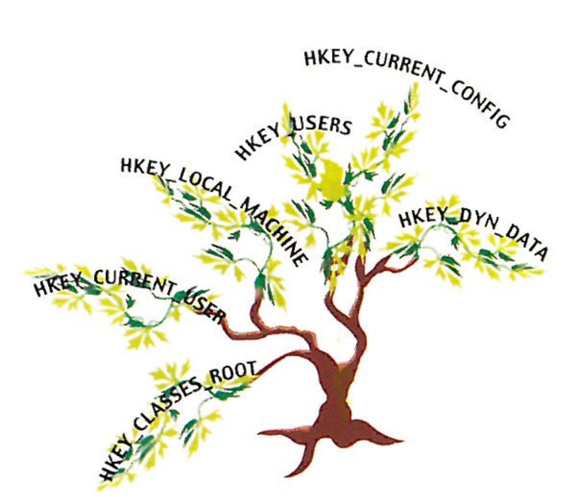
CONTROLANDO LOS ERRORES

Una hoja *XSL* puede dar lugar a dos tipos de errores: errores cuando el código *XSL* está siendo "parseado" o errores en tiempo de ejecución. Existe un objeto denominado *parseError* que puede ser consultado para saber si se ha producido algún error al "parsear" un documento *XML* o una hoja de estilo *XSL*. Cuando una fuente de datos *XML* se incluye en una página *HTML* mediante el uso de la etiqueta *<XML>* los errores que se producen durante el proceso de *parsing* no se indican en el navegador de forma alguna.

```
function iniciar() {
  if (documentoXML.parseError.
    errorCode != 0) {
    // Error en el documento XML
    mostrarErrorParser
    (documentoXML.parseError);
  } else {
    if (documentoXSL.parseError.
    errorCode != 0) {
      // Error en el documento
      // XSL
      mostrarErrorParser
      (documentoXML.parseError);
    } else {
      try {
        tabla.innerHTML =
      documentoXML.transformNode
      (documentoXSL.XMLDocument);
      } catch (exception) {
        // Error de ejecución

     mostrarErrorEjecucion
     (exception);
      }
      gastoTotal();
    }
  }
}
```

Hemos modificado la función *iniciar* de forma que controlamos si se producen errores a la hora de cargar tanto el documento *XML* como la hoja *XSL*. Los errores de ejecución estándar se muestran en el navegador con ventanas de alerta. Si queremos evitar esto podemos utilizar la sentencia *try...catch* que proporciona *Internet Explorer 5.0* para manejar excepciones.



Programación del registro de Windows (II)

Adolfo Aladro (aaladro@arrakis.es)

Continuamos con las funciones de la *API* de *Windows* que permiten acceder al registro. También iremos analizando algunos aspectos más concretos, lo que nos permitirá poner en práctica todas las llamadas de la *API* que vayamos abordando.

RECABAR INFORMACIÓN DEL REGISTRO

La *API* de *Windows* ofrece la posibilidad de recabar información del registro de diversas f ormas. Dependiendo de la aplicación que estemos desarrollando interesará una u otra. Por lo general el caso más frecuente es aquel en el que solamente leemos y/o escribimos una clave concreta del registro.

Pero a veces también es necesario conocer toda la información con respecto a una clave y todos los datos que de ella dependen: otras claves y los valores. Las funciones *RegQueryInfoKey*, *RegEnumKey*, *RegEnumKeyEx* y *RegEnumValue* facilitan la tarea de navegar por el registro en busca de la información deseada.

OBTENER INFORMACIÓN DE UNA CLAVE (REGQUERYINFOKEY)

Dado un identificador de clave podemos obtener información sobre dicha clave utilizando la función *RegQueryInfoKey*. Esta información se compone fundamentalmente del número de subclaves existentes, valor de longitud máximo entre las cadenas que representan los nombres de subclaves existentes, número de valores, valor de longitud máximo entre las cadenas que representan a los nombres los valores y valor de longitud máximo entre las cadenas que representan los contenidos y los valores.

```
Declare Function RegQueryInfoKey
    Lib "advapi32.dll"
    Alias "RegQueryInfoKeyA"
    (ByVal hKey As Long,
     ByVal lpClass As String,
     lpcbClass As Long,
     ByVal lpReserved As Long,
     lpcSubKeys As Long,
     lpcbMaxSubKeyLen As Long,
     lpcbMaxClassLen As Long,
     lpcValues As Long,
     lpcbMaxValueNameLen As Long,
     lpcbMaxValueLen As Long,
     lpcbSecurityDescriptor As Long,
     lpftLastWriteTime As FILETIME)
    As Long
```



- *HKey*
Identificador de la clave de la que queremos obtener información.
- *lpClass*
Variable donde se almacenará una cadena de texto que se corresponde con el nombre de la clase de la clave. Éste fue establecido en el momento de la creación. Este parámetro puede ser **NULL** (lo que en *Visual Basic* equivale a **0&**)
- *lpcbClass*
Número entero que indica el espacio disponible en la variable anterior. Si en el anterior se ha puesto **NULL** en este caso debe ser también **NULL**.
- *lpReserved*
Parámetro reservado (debe dejarse siempre con valor igual a **0**).
- *lpcSubKeys*
Variable donde se almacenará el número de subclaves existentes.
- *lpcbMaxSubKeyLen*
Variable donde se almacenará la longitud máxima de los nombres de subclaves existentes.
- *lpcbMaxClassLen*
Variable donde se almacenará la longitud máxima de los nombres de clase de cada subclave.
- *lpcValues*
Variable donde se almacenará el número de valores existentes.
- *lpcbMaxValueNameLen*
Variable donde se almacenará la longitud máxima de los nombres de los valores existentes.
- *lpcbMaxValueLen*
Variable donde se almacenará la longitud máxima del contenido de los valores.
- *lpcbSecurityDescriptor*
Variable donde se almacena el tamaño del descriptor de seguridad asociado a la clave.
- *lpftLastWriteTime*
Variable donde se almacena la fecha de la última modificación hecha a la clave.

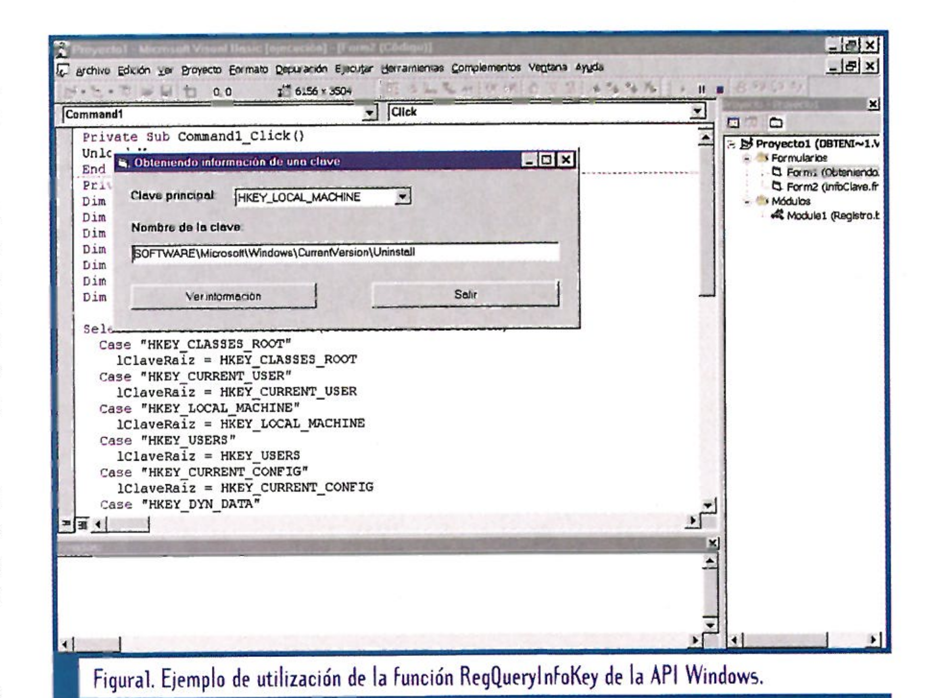


Figura1. Ejemplo de utilización de la función RegQueryInfoKey de la API Windows.

Esta función será muy útil como paso previo, antes de recorrer las ramas o los valores de una clave, ya que nos proporciona el número de estos elementos así como el espacio que necesitamos para almacenar los nombres, contenidos, etc.

Pasemos a realizar un primer ejemplo que utilice esta llamada a la *API* de *Windows*. El fichero **ObteniendoInfoClave.vbp** contiene el proyecto de *Visual Basic* con nuestra pequeña aplicación. Se trata de un programa sencillo en el que seleccionamos la clave principal, escribimos el nombre de la rama y tras pulsar el botón **Ver información** se muestra toda la información disponible acerca de esta clave del registro, si es que existe. El listado 1, que aparece en el CD-ROM, se muestra la rutina principal de la aplicación, que se ejecuta justo cuando se carga el segundo formulario, que es el que muestra la información.

En primer lugar tomamos la clave principal seleccionada por el usuario en la lista desplegable y el nombre de la clave del campo de texto. Una vez que disponemos de estos datos tenemos que intentar abrir la clave utilizando la llamada *RegOpenKeyEx*. Como ya vimos, esta función devuelve *ERROR_SUCCESS* si todo ha ido bien o un código de error en caso contrario.

En el caso de que la lectura haya finalizado con éxito estamos en disposición de obtener la información de la clave llamando a la función *RegQueryInfoKey*. Si no se produce ningún error obteniendo la información de la clave la función devolverá *ERROR_SUCCESS* y en ese instante actualizamos las etiquetas del formulario para mostrar los datos obtenidos.

Nótese que por defecto la aplicación se inicializa con la clave principal *HKEY_LOCAL_MACHINE* seleccionada y el nombre de la clave igual a **SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall**. Tal y como vimos en la entrega anterior, aquí es donde se guarda, en el registro, la información pertinente a la desinstalación de las

aplicaciones de *Windows,* que al instalarse han establecido así esta posibilidad.

RECORRER LAS RAMAS DE UNA CLAVE (REGENUMKEY Y REGENUMKEYEX)

La función *RegQueryInfoKey* permite averiguar el número de ramas de una clave. Teniendo esta información ya estamos en disposición de recorrer todas las ramas.

```
Declare Function RegEnumKey
  Lib "advapi32.dll"
  Alias "RegEnumKeyA"
   (ByVal hKey As Long,
    ByVal dwIndex As Long,
    ByVal lpName As String,
    ByVal cbName As Long) As Long
```

- *hKey*
Identificador de la clave abierta.
- *dwIndex*
Índice de la rama que se desea consultar.
- *lpName*
En esta variable es donde se almacenará el nombre.
- *cbName*
Número entero que expresa el espacio disponible en la cadena anterior.

Es importante tener en cuenta que el índice de las ramas irá desde cero hasta el número de ramas menos uno. En cualquier caso, no es necesario conocer a priori el número de ramas porque es posible invocar la función *RegEnumKey* indefenidamente hasta obtener *ERROR_NO_MORE_ITEMS*.

La *API* de *Windows* proporciona otra función, muy parecida a explicada anteriormente, con la que podemos recorrer las ramas de una clave obteniendo algo más de información.

```
Declare Function RegEnumKeyEx
  Lib "advapi32.dll"
  Alias "RegEnumKeyExA"
   (ByVal hKey As Long,
    ByVal dwIndex As Long,
    ByVal lpName As String,
    lpcbName As Long,
    ByVal lpReserved As Long,
    ByVal lpClass As String,
    lpcbClass As Long,
    lpftLastWriteTime As FILETIME)
  As Long
```

- *hKey*
Identificador de la clave abierta.
- *dwIndex*
Índice de la rama que se desea consultar.
- *lpName*
Aquí se almacenará el nombre.
- *cbName*
Número entero que expresa el espacio disponible en la cadena anterior.
- *lpReserved*
Parámetro reservado (debe dejarse siempre con valor igual a **0**).
- *lpClass*
Variable donde se almacenará el nombre de la clase.
- *lpcbClass*
Número entero que expresa el espacio disponible en la cadena anterior.
- *lpftLastWriteTime*
Variable donde se almacena la fecha de la última modificación hecha a la clave.

Tanto la variable *RegEnumKey* como la variable *RegEnumKeyEx* nos devuelven la sentencia *ERROR_SUCCESS* si todo ha salido bien o un código de error en caso contrario.

El fichero **EnumerarClaves.vbp** contiene un ejemplo de cómo podemos utilizar esta función para saber todas las claves inmediatas que surgen a partir de una rama del registro. Sin embargo, este programa es muy parecido al anterior, con la salvedad de que todas las claves encontradas se muestran en forma de lista.

El listado 2, situado en el CD-ROM, nos muestra el bucle principal. Como se puede observar se llama a la función *RegEnumKey* un número indefinido de veces hasta

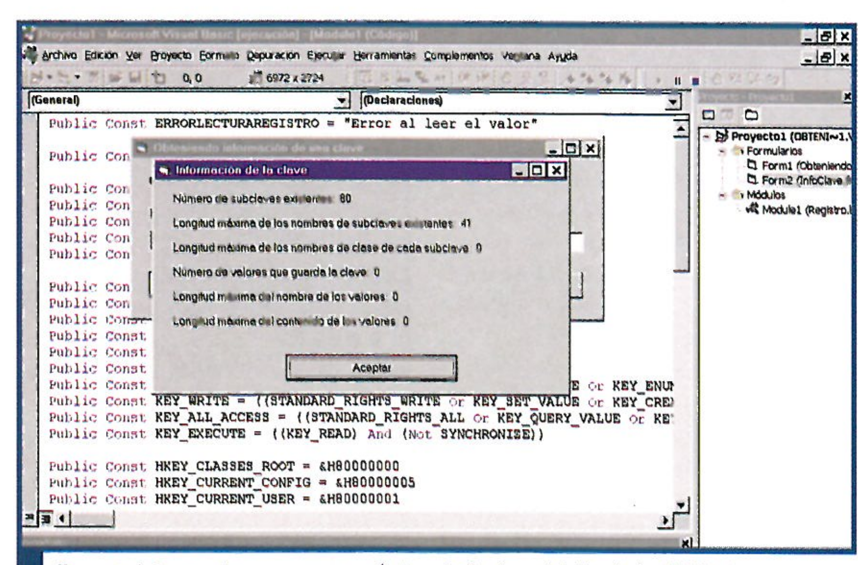


Figura 2. Información que proporciona la función RegQueryInfoKey de la API Windows.



que nos devuelve el valor *ERROR_NO_MORE_ITEMS*. Cada vez que encontramos una nueva clave la guardamos en la lista que ya tenemos. *InStr* devuelve un valor numérico que especifica la posición de la primera aparición de *Chr(0)* en la cadena correspondiente al nombre de la clave. Esta posición servirá para eliminar los espacios que no sirven a la derecha.

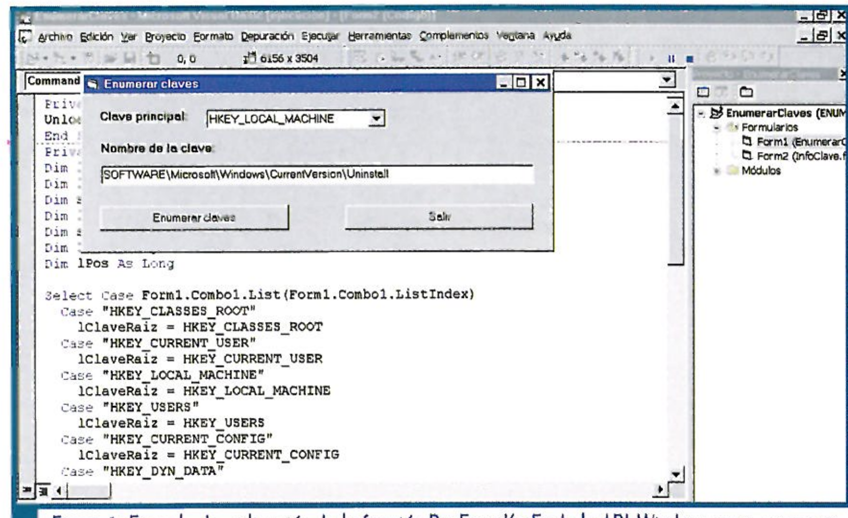


Figura 3. Ejemplo de utilización de la función RegEnumKeyEx de la API Windows para enumerar las subclaves de una clave.

RECORRER LOS VALORES DE UNA CLAVE (REGENUMVALUE)

Toda rama del registro además de contener otras ramas o subclaves también puede albergar valores. Todo valor tiene un nombre y un contenido. El contenido de un valor en el registro de *Windows* puede ser un número (*REG_DWORD*), una cadena (*REG_SZ*) o un valor binario (*REG_BINARY*). Existe un valor sin nombre, al que denominamos predeterminado, y que está presente en todas las claves.

Si queremos conocer todos los valores de una clave es conveniente haber obtenido la información de la clave, mediante la llamada a *RegQueryInfoKey*. De esta manera conoceremos de antemano el número de valores de esa clave, así como las longitudes máximas del nombre y el contenido de los valores existentes. De todas formas este procedimiento no es un requisito indispensable ya que es posible utilizar la misma técnica de nuestro ejemplo anterior para recorrer todos los valores. Es decir, llamamos a la función *RegEnumValue*, *también un número indefinido de vesces,* hasta que ésta devuelva *ERROR_NO_MORE_ITEMS*.

```
Declare Function RegEnumValue
  Lib "advapi32.dll"
  Alias "RegEnumValueA"
   (ByVal hKey As Long,
    ByVal dwIndex As Long,
    ByVal lpValueName As String,
    lpcbValueName As Long,
    lpReserved As Long,
    lpType As Long,
    lpData As Byte,
    lpcbData As Long) As Long
```

- *hKey*
Identificador de la clave abierta.
- *dwIndex*
Índice del valor que se desea consultar (debe estar entre 0 y el nº de valores menos uno)
- *lpValueName*
Variable donde se almacenará el nombre del valor.
- *lpcbValueName*
Número entero que expresa el espacio disponible en la cadena anterior.
- *lpReserved*
Parámetro reservado (siempre con valor igual a **0**).
- *lpType*
Variable numérica donde se almacenará el tipo del valor.
- *lpData*
Variable donde almacenará el contenido del valor.
- *lpcbData*
Número entero que indica el espacio disponible en la cadena anterior.

La API de Windows ofrece la posibilidad de recabar información del registro de diversas formas

En el caso de no estar interesados en conocer el tipo así como el contenido del valor, podemos omitir los tres últimos parámetros mencionados, para ello deberemos ponerlos a cero.

El ejemplo contenido en el proyecto **EnumerarValores.vbp** nos muestra cómo podemos conocer todos los valores de una rama del registro (el nombre del valor, su contenido y el tipo datos asociado).

El listado 3, que también se encuentra en el CD-ROM muestra el código del procedimiento princi-

pal de la aplicación. Cuando ésta arranca se muestra una lista desplegable con todas las carpetas especiales del sistema. Recordemos que éstas se correspondían con valores que da la clave

```
HKEY_CURRENT_USER\Software\
    Microsoft\Windows\CurrentVersion\
    Explorer\ShellFolders
```

Cada vez que hacemos *clic* en alguno de los elementos de la lista se muestran, en las casillas correspondientes, tanto el contenido como el tipo del valor. En el caso concreto de nuestro ejemplo estos valores serán la ruta completa donde se encuentra ese directorio especial y el tipo *REG_SZ,* ya que el contenido anterior es, evidentemente, una cadena de caracteres.

Para almacenar el contenido de los valores así como su tipo vamos a utilizar un par de matrices cuyo tamaño incrementaremos dinámicamente.

```
Valores() As String
Dim mTipos() As String
```

El procedimiento *ReDim Preserve* nos permite redimensionar el tamaño de una matriz. La palabra reservada *Preserve* se añade para significar que se desea que los datos originales de matriz se conserven.

Cada vez que aumentemos el tamaño de la matriz lo haremos en una unidad por lo que la variable **I** que sirve para llevar el bucle sirve perfectamente para esta labor.

Una vez que hemos abierto la clave y tenemos su identificador entramos en un bucle del que sólo salimos cuando *RegEnumValue* devuelve un valor diferente a *ERROR_SUCCESS*. Aunque ya hemos dicho que se pueden omitir los tres últimos parámetros, nosotros los vamos a utilizar todos.

El que llamará más la atención será el parámetro correspondiente al contenido del valor. Pasamos el primer elemento de un *array* de *bytes*. Ese *array* será el encargado de almacenar la cadena de caracteres con el contenido del valor. Evidentemente este *array* habremos de convertirlo en una cadena de caracteres antes de utilizarlo en nuestra aplicación. Este es el cometido que desempeña la función denominada *ArrayOfBytes ToString*. Por otro lado, la función *TipoDatos* nos ayuda a convertir el valor numérico correspondiente al tipo de datos en una cadena de caracteres, concretamente (*REG_SZ*, *REG_BINARY*, etc).

Figura 4. Lista de elementos formada por las subclaves que dependen de una clave dada.

```
Function TipoDatos(tipo As Long)
   As String
Select Case tipo
Case REG_BINARY
  TipoDatos = "REG_BINARY"
Case REG_DWORD
  TipoDatos = "REG_DWORD"
Case REG_DWORD_LITTLE_ENDIAN
  TipoDatos =
    "REG_DWORD_LITTLE_ENDIAN"
Case REG_DWORD_BIG_ENDIAN
  TipoDatos =
    "REG_DWORD_BIG_ENDIAN"
Case REG_EXPAND_SZ
  TipoDatos = "REG_EXPAND_SZ"
Case REG_LINK
  TipoDatos = "REG_LINK"
Case REG_MULTI_SZ
  TipoDatos = "REG_MULTI_SZ"
Case REG_NONE
  TipoDatos = "REG_NONE"
Case REG_RESOURCE_LIST
  TipoDatos = "REG_RESOURCE_LIST"
Case REG_SZ
  TipoDatos = "REG_SZ"
Case Else
  TipoDatos = "?"
End Select
End Function
```

Cada vez que el hacemos *clic* en alguno de los elementos de la lista consultamos nuestras matrices para ofrecer la información en las casillas correspondientes.

```
Private Sub List1_Click()
 If List1.ListIndex <> -1 Then
  Text1.Text =
    mValores(List1.ListIndex)
  Text2.Text =
  mTipos(List1.ListIndex)
 End If
End Sub
```

MÁS TRUCOS ACERCA DEL REGISTRO

ELIMINAR LA FRASE "ACCESO DIRECTO A"

Cuando creamos un acceso directo en *Windows* el sistema añade por defecto al nombre del acceso directo las palabras "**Acceso directo a**" (o *Shortcut to* si trabajamos con la versión inglesa). Es posible evitar esto



accediendo al registro de *Windows* y modificando:

```
HKEY_CURRENT_USER
    \Software\Microsoft\Windows\
    CurrentVersion\Explorer
```

Debemos buscar un valor llamado *link* y si no existe crearlo. Si hacemos que el valor sea igual a **00 00 00 00** desaparecerá el texto que se añade a los accesos directos.

CAMBIAR LA ALINEACIÓN POR DEFECTO DE LOS MENÚS DESPLEGABLES

Las opciones de los menús desplegables de *Windows* están alineadas a la izquierda por defecto. Podemos cambiarlo accediendo a la clave:

```
HKEY_CURRENT_USER\ControlPanel\
    Desktop
```

Debemos buscar el valor *MenuDropAlignment* y si no existe crearlo. Si hacemos ese valor igual a **1** se alineará a la derecha mientras que si es igual a **0** lo hará a la izquierda (el tipo de datos será *REG_SZ*).

CAMBIAR EL TAMAÑO DE LOS ICONOS DEL ESCRITORIO

Windows también nos proporciona una forma de cambiar el tamaño de los iconos del escritorio. Lo podemos hacer con esta orden:

```
HKEY_CURRENT_USER\ControlPanel\
    Desktop\WindowMetrics
```

Debemos buscar el valor *Shell Icon Size* y si no existe crearlo. Éste contendrá el tamaño de los iconos en *pixels*. El tipo de datos es *REG_SZ* y por defecto el valor es igual a **32**.

ACTIVAR Y/O DESACTIVAR EL AUTOARRANQUE DEL CD-ROM

Podemos activar y/o desactivar el autoarranque del *CD-ROM* accediendo a la clave:

```
HKEY_LOCAL_MACHINE\SYSTEM\
    CurrentControlSet\Services\CDRom
```

Debemos buscar el valor *Autorun* y si no existe crearlo. Si hacemos ese valor igual a **0** la función de autoarranque se desactivará, si es igual a **1** la función estará activa. El tipo de datos es *DWORD*.

CAMBIAR EL RETARDO CON EL QUE SE MUESTRAN LOS MENÚS

Windows normalmente muestra los menús con un cierto retardo de tiempo. Es posible modificar el valor correspondiente a ese retardo e incluso suprimirlo por completo.

```
HKEY_CURRENT_USER\ControlPanel\
    Desktop
```

Debemos buscar el valor *MenuShowDelay*. Éste almacena el número de milisegundos que tardan los menús en ser mostrados. El intervalo de valores puede oscilar entre **0** y **999**, y el tipo de datos asociado a este valor es *REG_SZ*.

CONTROLAR LA ANIMACIÓN DE LAS VENTANAS DE WINDOWS

La animación de las ventanas es una característica de *Windows* que resulta ciertamente atractiva pero que puede perjudicar el rendimiento de nuestro ordenador si éste no cuenta con una tarjeta gráfica demasiado potente.

```
HKEY_CURRENT_USER\ControlPanel\
    Desktop
```

Debemos buscar el valor *MinAnimate* y si no existe crearlo. Si hacemos ese valor igual a **0** la función de animación se desactivará y si es igual **1** se activará. El tipo de este valor es *REG_SZ*.

CONTROLAR EL SCROLLING DE WINDOWS

Al igual que ocurre con la característica anterior, es posible activar y/o desactivar la función de *smooth scrolling* de *Windows*.

```
HKEY_CURRENT_USER\ControlPanel\
    Desktop
```

Debemos buscar el valor *SmoothScroll* y si no existe crearlo. Si hacemos que sea igual a **00 00 00 00** la función se desactivará, y estará activa si es igual a **00 00 00 01**. El tipo de datos de este valor es *REG_BINARY*.

Figura 5. Ejemplo de utilización de la función RegEnumValue de la API Windows.

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (II)

Enrique de la Lastra (enriquelastra@hotmail.com)

El diseño del *software* para una comunicación punto a punto requiere de un componente para la correcta transmisión de datos binarios a través del puerto de comunicaciones. Para ello crearemos, en este artículo, un componente *Delphi*.

INTRODUCCIÓN

En el anterior artículo realizamos una introducción sobre el modelo de comunicaciones a seguir para llevar a cabo una correcta transferencia de información (ya fuese en forma de archivos o mensajes) entre dos ordenadores conectados directamente. Vimos que lo mejor era dividir todo el proceso en varias partes, funcionalmente superpuestas según el modelo de comunicaciones *OSI* (*Open Systems Interconnection,* Interconexión de Sistemas Abiertos). Nos aprovecharemos de las posibilidades de *Delphi* para crear un componente por cada nivel de comunicaciones de *OSI*. Además, gracias a la herencia, aprovecharemos el servicio que ofrece cada nivel para heredar cada componente del anterior en la jerarquía.

La primera de las partes funcionales debía estar formada por el código necesario para enviar datos al puerto (serie) de comunicaciones, para recibirlos en el otro extremo y para controlar el proceso. Para ello, implementaremos un componente *Delphi*, que realice estas funciones y que ofrezca un servicio al componente "hijo" independiente del puerto de comunicaciones.

DEFINICIÓN DE PROPIEDADES Y VARIABLES

Anteriormente se estudiaron las funciones de la *API* de *Windows CreateFile, GetCommState y SetCommState.* La primera de ellas abre el puerto de comunicaciones y devuelve un *Handle* al mismo. Las otras dos permiten configurar el puerto mediante una estructura de datos, denominada *DCB* (*Device Control Block,* Bloque de Control del Dispositivo).

Puesto que el *Handle* que devuelve la función *CreateFile* identifica unívocamente al puerto y es el que se utiliza en todas las llamadas a la *API* de *Windows*, crearemos una variable que almacene ese valor. Esa variable será privada, pues queremos que el nivel superior utilice este componente de forma transparente a la *API* de comunicaciones, y por tanto si quiere hacer referencia a un puerto, lo hará a través de una variable más significativa, como por ejemplo un *string* del tipo **"COM1"**.



Figura 1: Comunicación lógica y comunicación real que se realiza entre dos máquinas.

También crearemos una variable privada de tipo *TDCB* que utilizaremos para actualizar las variables de la estructura *DCB* y para configurar el puerto de comunicaciones:

```
private
  PortHandle: THANDLE;

  DCB: TDCB;
```

El Handle que devuelve la función CreateFile identifica unívocamente al puerto

Como vamos a crear un componente *Delphi*, heredado de *TComponent* y que por tanto, se podrán editar sus propiedades desde el *Object Inspector*, crearemos tantas propiedades como variables significativas contiene la estructura *DCB*. En concreto las variables que haremos públicas en el *Object Inspector* serán:

- *BaudRate*: Velocidad de transferencia en *bits* por segundo.
- *ByteSize*: Número de *bits* por cada *byte* de datos
- *Parity*: Tipo de paridad.
- *StopBits*: *Bits* de parada.
- *FlowControl*: Tipo de control de flujo (tanto *Hardware* como *Software*: tanto *RTS – CTS*, como *XON-XOFF*).

Debido a que serán propiedades editables por el usuario desde el *Object Inspector* y pensando en un uso aislado del componente, definiremos unos tipos enumerados de datos que recojan todos los valores posibles que pueden asumir estas variables. De esta forma su edición desde el *Object Inspector* se reducirá a seleccionar un elemento en un control desplegable.

Teniendo en cuenta los posibles valores de cada una de estas variables, los nuevos tipos de datos serán los siguientes:

```
{ Tasas Binarias del puerto
    Serie: 110 bps, 300 bps ,
    600 bps , 1200 bps , 2400
    bps , 4800 bps , 9600 bps ,
    14400 bps , 19200 bps ,
    38400 bps , 56000 bps, 57600
    bps, 115200 bps, 128000 bps,
    256000 bps }

TBaudRate =
(br110,
br300,
br600,
br1200,
br2400,
br4800,
br9600,
br14400,
br19200,
br38400,
br56000,
br57600,
br115200,
br128000,
br256000);

{ Número de bits de datos por
    byte: 4, 5, 6, 7 u 8}
TByteSize =
(bsFour,
bsFive,
bsSix,
bsSeven,
bsEight);

{ Tipo de paridad: Ninguna,
    Impar, Par, Marca, Espacio }
TParity =
(pbNone,
pbOdd,
pbEven,
pbMark,
pbSpace);

{ Bits de parada: 1 bit, 1.5
    bits, 2 bits }
TStopBits =
(sbOneStop,
sbOneFiveStop,
sbTwoStop);

{ Control de flujo: sin control
    de flujo, control de flujo
    Hardware, control de flujo
    Software o ambos }
TFlowControl =
(fcNone,
fcRTSCTS,
fcXONXOFF,
fcBoth);
```

Además habrá que publicar también una propiedad que contenga el puerto serie (*COM*) por el que queremos realizar la comunicación y otras dos con los tamaños de los *buffers* de transmisión y recepción de datos (es decir de salida y entrada de datos). A la primera la denominaremos *PortNumber* y a los *buffers*: *BufferTxSize* y *BufferRxSize*. Para la propiedad que identifica el puerto también vamos a definir un nuevo tipo enumerado de forma que sus valores queden delimitados:

```
{ Número de puerto serie }
TPortNumber =
(COM1,
```

```
COM2,
COM3,
COM4);
```

Una vez definidas las propiedades públicas, pasaremos a codificarlas. Como en ninguna de ellas queremos realizar una función al leer su valor, sólo tendrán asociadas un procedimiento de escritura del tipo *SetProperty* en el que realizaremos la actualización de la estructura *DCB* y a su vez reconfiguraremos el puerto de comunicaciones.

Crearemos tantas propiedades como variables importantes ofrece la estructura DCB

El código que generaremos en la clase, para definir estas variables y propiedades será el siguiente (recordemos del pasado artículo que pusimos por nombre al componente *TComSerie*):

```
TComSerie = class(TComponent)
private
 { Handle al puerto de comunica-
    ciones serie }
  PortHandle: THANDLE;
 { Estructura de datos del puerto
    }
  DCB: TDCB;

protected
  FPortNumber : TPortNumber;
  FBaudRate   : TBaudRate;
  FByteSize   : TByteSize;
  FParity     : TParity;
  FStopBits   : TStopBits;
  FFlowControl: TFlowControl;
  FBufferRxSize : word;
  FBufferTxSize : word;

  procedure SetPortNumber
    (AValue: TPortNumber);
  procedure SetBaudRate
    (AValue: TBaudRate);
  procedure SetByteSize
    (AValue: TByteSize);
  procedure SetParity
    (AValue: TParity);
  procedure SetStopBits
    (AValue: TStopBits);
  procedure SetFlowControl
    (AValue: TFlowControl);
  procedure SetBufferRxSize
    (AValue: word);
  procedure SetBufferTxSize
    (AValue: word);

published
  property PortNumber:
    TPortNumber read FPortNumber
    write SetPortNumber;
 property BaudRate: TBaudRate
  read FBaudRate write
    SetBaudRate;
 property ByteSize: TbyteSize
  read FByteSize write
    SetByteSize;
 property StopBits: TStopBits
  read FStopBits write
    SetStopBits;
 property Parity: TParity
  read FParity   write SetParity;
 property FlowControl:
    TFlowControl
  read FFlowControl
  write SetFlowControl;
 property BufferRxSize: word
  read FBufferRxSize
  write SetBufferRxSize;
 property BufferTxSize: word
  read FBufferTxSize
  write SetBufferTxSize;
end; { Definición de la clase }
```

PINES EN UN CONECTOR DB25

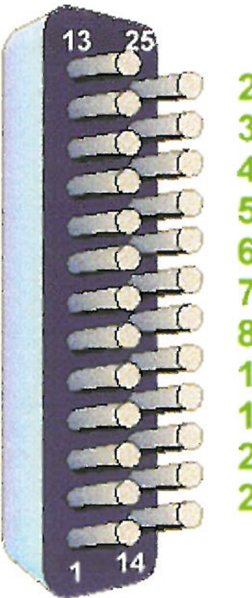


Figura 2: Distribución de pins en un conector DB25 con interfaz R5232

APERTURA Y CONFIGURACIÓN DEL PUERTO DE COMUNICACIONES

Con las variables definidas y las funciones suministradas ya podemos generar el código del procedimiento que se encargará de abrir el puerto seleccionado y con las características requeridas.

Para ello, debemos abrir el puerto con la función *CreateFile*,

asignando las constantes adecuadas a los parámetros (según vimos en el anterior artículo) y convirtiendo el número de puerto (que vendrá dado en forma de enumerado a través de la propiedad *FPortNumber*) en una cadena de caracteres (como por ejemplo **"COM1"**) terminada en nulo.

El Handle lo almacenaremos en una propiedad privada de la clase: PortHandle

El valor devuelto por la función lo recuperaremos en la variable *PortHandle*. Si este valor es **0**, significa que el puerto no se ha podido abrir y en cualquier otro caso será un valor que referencia unívocamente al puerto abierto.

Con un valor válido de *PortHandle* realizamos una llamada a la función *GetCommState,* lo que servirá para rellenar todas las variables de la estructura *DCB* con valores válidos. Después asignaremos los valores de las propiedades de la clase relacionadas con *DCB*, a sus variables correspondientes. Y una vez hecho esto, realizaremos una llamada a la función de la *API SetCommState* para que configure el puerto de comunicaciones con los valores de esta estructura.

Como la función *SetCommState* configura todas las variables del puerto, exceptuando los *buffers* de entrada y salida de datos, emplearemos otra función de la *API*, *SetupComm,* que se encarga de esta función. Antes de ello vaciaremos ambos *buffers* con unos procedimientos que crearemos para ese propósito. El código de la función que abre el puerto y lo configura, quedará tal y como se muestra en el Listado 1.

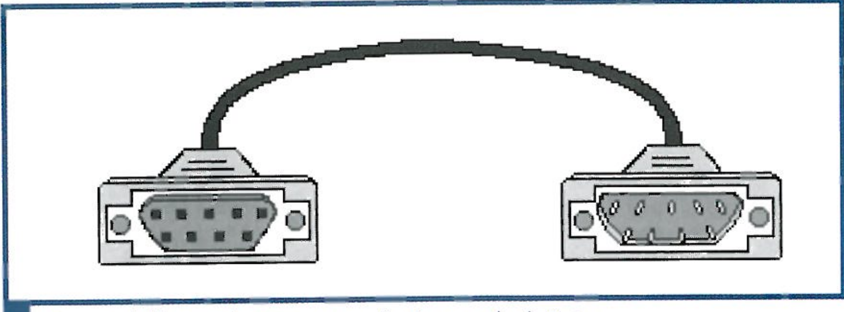


Figura 3: Cable cruzado con conectores hembra y macho de 9 pins.

Listado 1: Código del procedimiento AbrirPuerto, encargado de abrir y configurar un puerto serie.

```
function TComSerie.AbrirPuerto: boolean;
var
  NumCOM: array[0..4] of char;

begin
  Result := True;
  if PortHandle > 0 then exit;

  StrPCopy (NumCOM, 'COM' + chr(ord(FPortNumber) + ord('1')));

  PortHandle := CreateFile
    ( NumCOM,
     GENERIC_READ or GENERIC_WRITE,
     0,
     nil,
     OPEN_EXISTING,
     FILE_ATTRIBUTE_NORMAL,
     0);

  if (PortHandle = 0) then begin
    Result := False;
    exit;
  end;

  GetCommState (PortHandle, DCB);
  DCB.BaudRate := ObtainDCB_BaudRate (FBaudRate);
  DCB.ByteSize := ord(FByteSize) + 4;
  DCB.Parity := ord(FParity);
  DCB.StopBits := ord(FStopBits);
  DCB.Flags := ObtainDCB_Flags (DCB.Flags, FFlowControl);

  SetCommState (PortHandle, DCB);

  FlushBufferRx;
  FlushBufferTx;
  SetupComm (PortHandle, FBufferRxSize, FBufferTxSize);

end; { AbrirPuerto }
```

FUNCIONES AUXILIARES

Según hemos visto en el código anterior, hay una llamada a varias funciones auxiliares. Dos de ellas: *ObtainDCB_BaudRate* y *ObtainDCB_Flags*, sirven para traducir los valores asignados a la variable *BaudRate* y a los *Flags*. En realidad esta última sólo asigna los valores relacionados con la propiedad *FlowControl*, pero está diseñada para que sea fácilmente ampliable al resto de los *Flags* que se quieran asignar.

La función *ObtainDCB_BaudRate* convierte el enumerado *TBaudRate* en una de las constantes predefinidas para los posibles valores de la variable *BaudRate* de *DCB*, como se puede ver en el siguiente código:

```
function TComSerie.ObtainDCB_Bau-
    dRate (ABaudRate: TBaudRate):
    DWORD;
begin
  case ABaudRate of
    br110:      Result := CBR_110;
    br300:      Result := CBR_300;
    br600:      Result := CBR_600;
    br1200:     Result := CBR_1200;
    br2400:     Result := CBR_2400;
    br4800:     Result := CBR_4800;
    br9600:     Result := CBR_9600;
    br14400:    Result :=
    CBR_14400;
    br19200:    Result :=
    CBR_19200;
    br38400:    Result :=
    CBR_38400;
    br56000:    Result :=
    CBR_56000;
    br57600:    Result :=
    CBR_57600;
    br128000: Result :=
    CBR_128000;
    br256000: Result :=
    CBR_256000;
  else          Result := 0;
  end; {case}
end; { ObtainDCB_BaudRate }
```

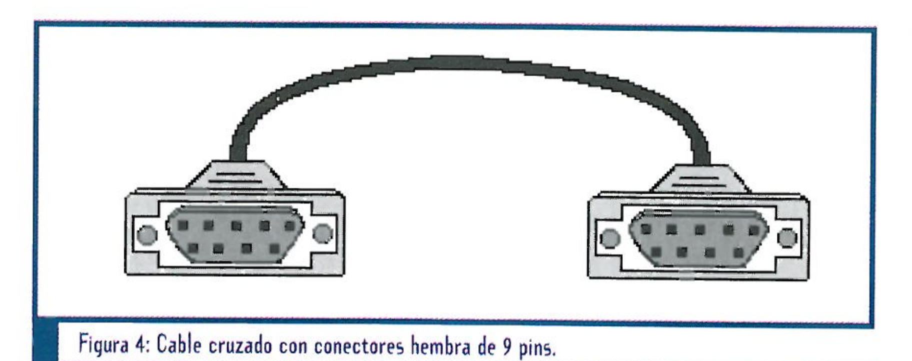


Figura 4: Cable cruzado con conectores hembra de 9 pins.

La función *ObtainDCB_Flags* utiliza el valor seleccionado en *TFlowControl* para asignar los valores a los *Flags* correspondientes. Para rellenar los *Flags* debemos tener en cuenta que cada uno de ellos está formado por un número de *bits*, tal y como se indica en la definición de la estructura (ver artículo anterior). Por ejemplo el primer *Flag*, *fBinary* se define como:

```
DWORD fBinary: 1
```

Lo cual indica que utiliza un *bit* para definir su valor. Por tanto puede valer **1** ó **0**. Además cada *Flag* ocupa un *bit* diferente en la estructura, por lo que al estar definidos los *Flags* en *Delphi* como un *longint* (8 *bytes*), si queremos asignar un valor a uno de los *Flags* mediante un *longint* lo mejor será definir unas constantes que tomen los valores deseados para cada *bit*.

Definimos unas constantes que predeterminen los posibles valores de los Flags de DCB

Para ello definimos unas constantes con los valores binarios adecuados (utilizando hexadecimal, que es más fácil):

```
dcb_Binary = $00000001;
dcb_Parity = $00000002;
dcb_OutxCtsFlow = $00000004;
dcb_OutxDsrFlow = $00000008;
dcb_DtrControlDisable =
    $00000000;
dcb_DtrControlEnable =
    $00000010;
dcb_DtrControlHandshake =
    $00000020;
dcb_DsrSensivity = $00000040;
dcb_TXContinueOnXoff =
    $00000080;
dcb_OutX = $00000100;
dcb_InX = $00000200;
dcb_ErrorChar = $00000400;
dcb_Null = $00000800;
dcb_RtsControlDisable =
    $00000000;
dcb_RtsControlEnable =
    $00001000;
dcb_RtsControlHandshake =
    $00002000;
dcb_AbortOnError = $00004000;
dcb_Dummy2 = $FFFF8000;
```

Teniendo en cuenta que la *API Win32* sólo soporta el modo binario, el *bit* correspondiente a *fBinary* debe ser **1**, para lo cual definimos la constante *dcb_Binary* que vale **1** en el *bit* adecuado.

Utilizando estas constantes, podemos asignar los valores relacionados con el control de flujo. Si queremos un control de flujo *Hardware* (*RTS – CTS*), añadiremos a los *Flags* (mediante un *"or"*) las constantes correspondientes a activar el control de la señal *CTS* (*dcb_OutxCtsFlow*) y a permitir el

intercambio de señales *DTR* (*dcb_RtsControlHandshake*). En el caso del control de flujo *software* (mediante señales *XON y XOFF*), para activarlo habrá que añadir las constantes *dcb_OutX* (que especifican el uso del control de flujo durante la transmisión) y *dcb_InX* (igual, en la recepción).

NOTA: Para más información sobre el control Hardware y el control Software se puede buscar en la ayuda On-line de la API de Windows.

Siguiendo estos criterios, el procedimiento quedaría como sigue a continuación:

```
function TComSerie.ObtainDCB_Flags
    (AFlags: longint;
    AFlowControl: TFlowControl):
    longint;
begin
  AFlags := AFlags or dcb_Binary;
  AFlags := AFlags and
    (not(dcb_OutxCtsFlow or
    dcb_RtsControlHandshake or
    dcb_OutX or dcb_InX));
  case AFlowControl of
  fcNone:    ;
  fcRTSCTS: AFlags:= AFlags or
    dcb_OutxCtsFlow or
    dcb_RtsControlHandshake;
  fcXONXOFF: AFlags := AFlags or
    dcb_OutX or dcb_InX;
  fcBoth: AFlags:= AFlags or
   (dcb_OutxCtsFlow or
    dcb_RtsControlHandshake) or
   (dcb_OutX or dcb_InX);
  end;
  Result := AFlags;
end; { ObtainDCB_Flags }
```

Por último reseñar otros dos procedimientos que utilizamos en la función *AbrirPuerto* (ver listado 1). Estos son *FlushBufferRx* y *FlushBufferTx* que permiten vaciar el *buffer* de entrada y el *buffer* de salida.

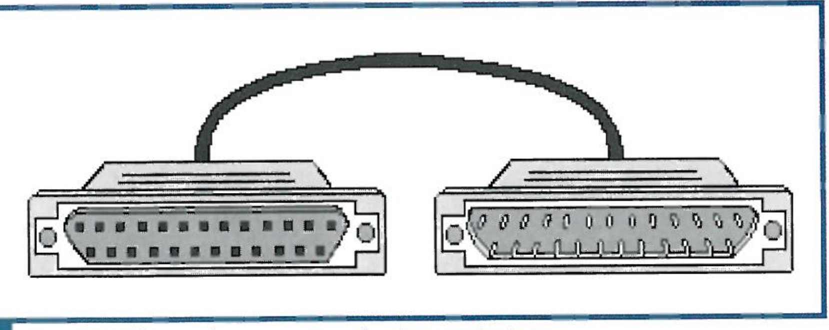
Para vaciar un *buffer* se utiliza la función de la *API PurgeComm*, que toma dos parámetros: el *Handle* al puerto y la acción a realizar sobre el puerto. Las posibles acciones vienen definidas por cuatro constantes, a saber:

- *PURGE_TXABORT*: Termina todas las operaciones de escritura pendientes, aunque no hayan terminado.
- *PURGE_RXABORT*: Igual que el anterior, pero en *las operaciones de lectura*.
- *PURGE_TXCLEAR*: Limpia el *buffer de salida*.
- *PURGE_RXCLEAR*: Limpia el *buffer* de entrada.

Por tanto, como los procedimientos los vamos a definir para vaciar los *buffers*, y no para cancelar las transmisiones pendientes, el código será el siguiente:

```
procedure TComSerie.FlushBufferRx;
begin
  { Comprobamos que existe un
    Handle válido, es decir que
    se ha abierto un puerto,
    antes de proceder }
  if PortHandle > 0 then
    PurgeComm (PortHandle,
    PURGE_RXCLEAR);
end; { FlushBufferRx }

procedure TComSerie.FlushBufferTx;
begin
  if PortHandle > 0 then
    PurgeComm (PortHandle,
    PURGE_TXCLEAR);
end; { FlushBufferTx }
```


Figura 5: Cable cruzado con conectores hembra y macho de 9 pins.

PROCEDIMIENTOS DE ESCRITURA DE LAS PROPIEDADES

Hasta aquí hemos visto lo fundamental para abrir el puerto de comunicaciones y configurarlo. Sólo falta desarrollar el código que asigna el valor a las variables internas, cuando el usuario del componente cambia una propiedad.

Los que modifican una variable de la estructura *DCB* son muy simples, ya que sólo requieren obtener la estructura *DCB* actual, asignar el valor y reconfigurar el puerto con el nuevo valor. Por ejemplo, el procedimiento *SetBaudRate* tendrá la siguiente forma:

```
procedure TComSerie.SetBaudRate
    (AValue: TBaudRate);
begin
  FBaudRate := AValue;
  if PortHandle > 0 then begin
    GetCommState (PortHandle,
    DCB);
    DCB.BaudRate :=
      ObtainDCB_BaudRate
     (AValue);
    SetCommState (PortHandle,
    DCB);
  end; { if }
end; { SetBaudRate }
```

El resto de procedimientos para las propiedades asociadas a variables de la estructura *DCB* siguen el mismo patrón (*SetByteSize*, *SetParity*, *SetStopBits*, *SetFlowControl*).

En cuanto al resto de métodos (*SetPortNumber*, *SetBufferRxSize* y *SetBufferTxSize*), su única función es asignar el nuevo valor, pero comprobando previamente que el puerto está cerrado (ya que sólo se pueden asignar al abrir el puerto).

```
procedure TComSerie.SetPortNumber
     (AValue: TPortNumber);
begin
  if PortHandle = 0 then
   FPortNumber := AValue;
end; { SetPortNumber }

procedure TComSerie.
     SetBufferRxSize
     (AValue: word);
begin
  if PortHandle = 0 then
   FBufferRxSize := AValue;
end; { SetBufferRxSize }

procedure TComSerie.
     SetBufferTxSize
     (AValue: word);
begin
  if PortHandle = 0 then
   FBufferTxSize := AValue;
end; { SetBufferTxSize }
```

ENVÍO DE DATOS

Para enviar datos al puerto de comunicaciones, nuevamente la *API* pone a nuestra disposición una función *WriteFile*, que se encarga de escribir un *buffer* de datos en el puerto de comunicaciones abierto. Como en el resto de las funciones de la *API*, el primer parámetro es el *Handle* al puerto, al que siguen los datos, su número, un parámetro de retorno del número de *bytes* enviados y un puntero a una estructura *OVERLAPPED* (sólo útil si estamos desarrollando una aplicación *multithreading*, que no es el caso). El valor devuelto es un *boolean* que indica si la operación ha tenido éxito.

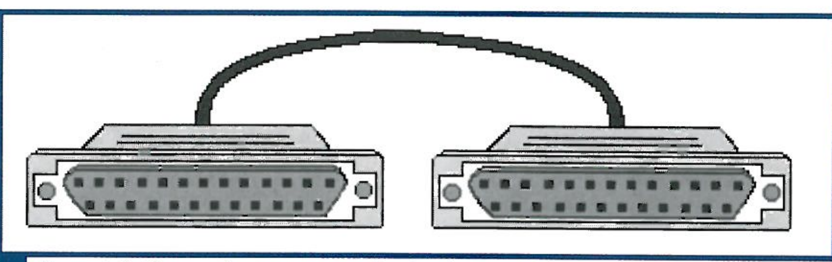


Figura 6: Cable cruzado con conectores hembra de 25 pins. Este es el más común para conectar dos PC's, pues los de 9 pins suelen ser utilizados para el ratón.

Invocaremos a la función WriteFile para escribir datos en el puerto

La única comprobación que debemos realizar, aparte del valor devuelto, es que el número de *bytes* efectivamente enviados coincide con el número de *bytes* que se le entregó a la función *WriteFile*:

```
function TComSerie.EnviarDatos
     (pDatos: pointer; NumDatos:
     integer): boolean;
var Enviados: DWORD;
begin
 FlushBufferTx;
 if WriteFile (PortHandle,
   pDatos^, NumDatos, Enviados,
   nil) then
  begin
   if (Enviados = DWORD(NumDa-
    tos))
    then Result := True
   else
    Result := False;
  end
 else
   Result := False;
end; { EnviarDatos }
```

En una versión más depurada, deberíamos realizar un bucle en el que, si no se enviaron todos los datos, realizaríamos tantas llamadas como fuera necesario a la función *WriteFile*.

RECEPCIÓN DE DATOS

El proceso de recepción de los datos es algo más complejo, ya que idealmente bastaría con un evento que avisase de la llegada de datos al puerto de comunicaciones. Lamentablemente, este evento no existe, y por tanto es necesario definir uno que permita comprobar si hay datos en el puerto de comunicaciones y en ese caso generar un nuevo evento que, esta vez sí, avise al usuario del componente de la llegada de datos al puerto serie.

El evento que utilizaremos será, claro está, un *Timer*. De esta manera comprobaremos de forma periódica el puerto de comunicaciones y cuando veamos que hay datos, generaremos nuestro evento de "datos recibidos".

No existe ningún mensaje de Windows que avise de la llegada de datos al puerto

Para definir un *Timer* tenemos la función de la *API SetTimer*. El problema de esta función es que requiere una ventana de *Windows* a la que enviar los mensajes, por lo

que previamente deberemos crear el *Handle* a esa ventana "fantasma".

Para ello disponemos de la función *AllocateHWnd*, que crea una ventana y tiene como único parámetro la función a la que se va a llamar cuando se reciban mensajes de *Windows*. Esta función tendrá la forma:

```
procedure WndProc
     (var msg: TMessage);
```

El código a generar será:

```
private
  FHWnd: HWND;
  procedure WndProc (
    var msg: TMessage);

...

constructor TComSerie.Create
     (AOwner: TComponent);
begin
  inherited Create (AOwner);
...
...
  { Comprobamos que no estamos en
    modo diseño }
  if not (csDesigning in
   ComponentState) then
    FHWnd := AllocateHWnd (Wnd-
    Proc);
end;
```

Debemos activar el *Timer* después de abrir el puerto de comunicaciones, que es cuando se comenzará a recibir información. Al final de la función *AbrirPuerto* se añadirá:

```
SetTimer (FHWnd, 1234, 50, nil);
```

El segundo parámetro de la función *SetTimer* es un identificador del *Timer*, el tercero la duración en milisegundos del intervalo entre eventos (**50** milisegundos en nuestro caso), y el cuarto una función a la que llamar cuando venza el intervalo. Si este último parámetro vale **Nil** entonces se genera un mensaje *WM_TIMER*. Esta será la forma de trabajo en nuestro caso.

Utilizaremos un Timer para controlar periódicamente el puerto

El procedimiento que está a la escucha de mensajes, y que hemos denominado *WndProc*, deberá comprobar, para cada mensaje que llega, si corresponde a *WM_TIMER* y en caso afirmativo leer la información que hay en el puerto de comunicaciones mediante la función de la *API ReadFile* (cuyos parámetros son equivalentes a los de *WriteFile*).

Como es posible que no haya datos en el puerto, el cuarto parámetro de esta función informará de si han llegado datos, hecho que se verificará cuando su valor sea distinto de **0**. Cuando esto ocurra generaremos un evento personalizado (que hemos denominado *OnDataReceived*). En él aparecerá la información de los datos que han llegado. De acuerdo con lo explicado, el procedimiento *WndProc* tendrá la siguiente estructura:

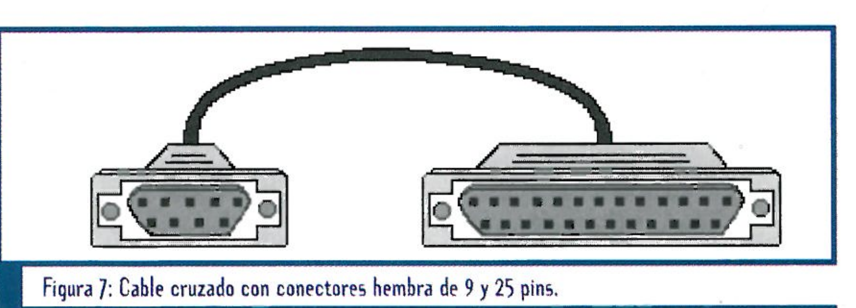
```
procedure TComSerie.WndProc
     (var msg: TMessage);

var Leidos: DWORD;
begin
  if (msg.Msg = WM_TIMER) and
    (PortHandle > 0) then begin
   {Leemos el buffer de Recepción
    }
   if ReadFile (PortHandle,
      BufferRx^, FBufferRxSize,
      Leidos, nil) then begin
      { Si se han leído caracte-
    res del puerto y se ha asig-
    nado el evento OnDataReceived
    en alguna instancia de la
    clase, disparamos el evento }
  if (Leidos <> 0) and
     Assigned(FOnDataReceived)
    then
       FOnDataReceived (Self,
         BufferRx, Leidos);
    end; { if }
  end; { if }
end; { WndProc }
```

CONCLUSIÓN

Hemos creado el código básico de un componente *Delphi* cuya función es transmitir y recibir datos a través de cualquier puerto serie del ordenador, aunque faltan pequeños matices. Terminaremos en el próximo artículo el componente, añadiendole algunas mejoras. También desarrollaremos una aplicación práctica que permitirá enviar y recibir mensajes de texto a través de un puerto serie (una especie de *chat* punto a punto).

Aunque la aplicación permitirá comunicar dos ordenadores a través del puerto serie, el componente sólo será el primer eslabón. Al final podremos transmitir, controlando y subsanando los errores de forma transparente, archivos y mensajes de texto entre dos máquinas conectadas punto a punto.


Figura 7: Cable cruzado con conectores hembra de 9 y 25 pins.

DIRECTX 6.1 (y IV)



Constantino Sánchez Ballesteros (constantino@nexo.es)

Finalizamos la serie de artículos relacionados con DirectX indagando en las posibilidades de DirectInput. Este componente nos permite dar soporte para el control de periféricos de entrada como pueden ser el ratón, el teclado, el joystick, e incluso el force-feedback.

INTRODUCCIÓN

DirectInput ofrece soporte para dispositivos de entrada como el ratón, teclado, *joystick* y otros controladores de juegos, incluyendo dispositivos del tipo *force-feedback*. Al igual que los demás componentes de *DirectX*, *DirectInput* se basa en el modelo *COM*.

Puesto que existen nuevos servicios para dispositivos no implementados por la *API Win32*, por ejemplo dispositivos *force-feedback*, *DirectInput* otorga un rápido acceso para la entrada de datos comunicándose directamente con los *drivers* del *hardware* en lugar del sistema de mensajes propios del sistema operativo *Windows*.

Los servicios extendidos y el rendimiento de *DirectInput* hacen de esta *API* una herramienta ideal para juegos, simulaciones y otras aplicaciones interactivas que requieran uso extensivo del control de dispositivos externos.

ARQUITECTURA DE DIRECTINPUT

La arquitectura básica de *DirectInput* consiste en un objeto *DirectInput*, que soporta una interfaz *COM* y un objeto por cada dispositivo de entrada que gestione datos. Cada dispositivo tiene instancias de objetos que son controles individuales o interruptores como teclas, botones o ejes. Los efectos individuales de la tecnología *force-feedback* también son representados por objectos.

NOTA: *la palabra "objeto" es utilizada para describir una entidad creada por el sistema DirectInput para soportar los métodos de una interfaz COM, siempre y cuando esos métodos no sean llamados a través de un lenguaje de programación orientado a objetos como C++. Para no entrar en confusiones, "objeto" significará para nosotros uno de los controles individuales o un dispositivo de entrada.*

En interés de la velocidad y respuesta, *DirectInput* trabaja directamente con los *drivers* de dispositivos, olvidando el sistema de mensajes de *Windows*. Además, *DirectInput* permite, a una aplicación, obtener acceso a los dispositivos de entrada aún cuando ésta no tenga el "foco" de la ventana (aplicación activa).

EL OBJETO DIRECTINPUT

Este objeto representa en una aplicación el subsistema *DirectInput*. Es utilizado para enumerar y manejar los dispositivos de entrada. Podemos crear el objeto *DirectInput* llamando a la función *DirectInputCreate*, que devuelve un puntero a una interfaz *COM IDirectInput*. Existen diferentes versiones



de esta interfaz para los *sets* de caracteres *ANSI* y *Unicode*.

Una vez creado el objeto *DirectInput*, podemos utilizar los métodos de la interfaz para enumerar los dispositivos individuales disponibles en el sistema y crear un objeto *DirectInputDevice* por cada dispositivo que deseemos utilizar en nuestra aplicación.

EL OBJETO DIRECTINPUTDEVICE

Cada objeto *DirectInputDevice* representa un dispositivo de entrada como el ratón, el teclado o el *joystick*. Podemos crear una instancia de un *DirectInputDevice* llamando al método *IDirectInput::CreateDevice*, que devuelve un puntero a la interfaz *IDirectInputDevice*. Los métodos de *IDirectInputDevice* son utilizados para obtener información sobre el dispositivo, establecer sus propiedades y obtener datos de los mismos.

NOTA: *Un dispositivo físico que es realmente una combinación de diferentes tipos de dispositivos de entrada, por ejemplo los ratones con trackball, pueden ser representados por dos o más objetos DirectInputDevice. Un dispositivo force-feedback es representado por un simple objeto joystick que gestiona tanto la entrada como la salida.*

DIRECTINPUT gestiona los periféricos de entrada para juegos (ratón, joystick, etc.)

INSTANCIAS DE DISPOSITIVOS DE OBJETOS DIRECTINPUT

Una *instancia objeto*, muchas veces llamada simplemente un *objeto*, es uno de los varios interruptores o controles disponibles en el dispositivo de entrada. Por ejemplo, instancias objeto de un *joystick* pueden incluir el eje X, eje Y, diversos botones y una palanca deslizante tipo *throttle*. Los objetos ratón pueden incluir dos o tres botones, el eje X, eje Y y una rueda. Para un teclado, cada tecla es un objeto.

La aplicación puede saber el número y tipo de objetos disponibles en un dispositivo a través del método *IdirectInputDevice::EnumObjects*. Los objetos individuales del dispositivo no son encapsulados como objetos código sino descritos como estructuras del tipo *DIDEVICEOBJECTINSTANCE*.

EL OBJETO DIRECTINPUTEFFECT

Un objeto de este tipo representa un efecto *force-feedback* que hemos definido. Es utilizado para manipular el efecto sobre la entrada/salida del dispositivo. Podemos crear un objeto *DirectInputEffect* llamamos al método *IdirectInputDevice2::CreateEffect*, que retorna un puntero a una interfaz *COM IDirectInputEffect*.

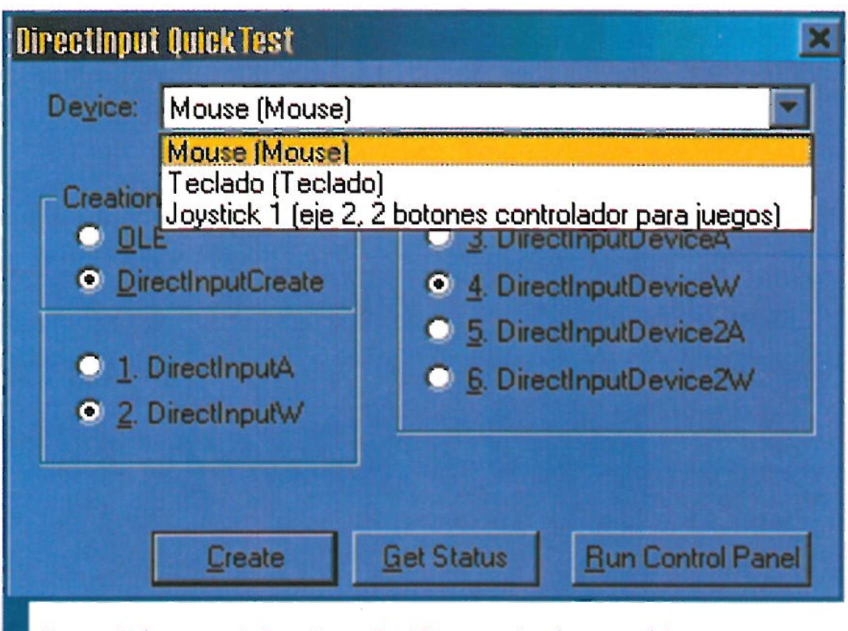


Figura 1: En la ventana de *DirectInpuct Quick Test* se puede seleccionar el dispositivo.

INTEGRACIÓN CON WINDOWS

Puesto que *DirectInput* trabaja directamente con los *drivers* del dispositivo, suprime o ignora los mensajes del ratón y teclado. Cuando estamos utilizando el ratón en modo exclusivo, *DirectInput* suprime los mensajes del ratón; como resultado, *Windows* es incapaz de mostrar el cursor en la pantalla.

DirectInput también ignora los valores asignados por el usuario al ratón y teclado mediante el panel de control. Para el teclado, la repetición de caracteres no es utilizada por *DirectInput*. Cuando utilizamos datos con *buffers*, *DirectInput* interpreta cada pulsación y liberación de tecla como un simple evento sin repeticiones. Cuando utilizamos datos inmediatos, a *DirectInput* sólo le concierne el estado físico presente de las teclas, no los eventos interpretados por *Windows*.

En el caso del ratón, *DirectInput* ignora los valores del panel de control como aceleración y cambio de botones.

NOTA: *Los valores establecidos en el driver por sí mismos serán reconocidos por DirectInput. Por ejemplo, si el usuario tiene un ratón de tres botones y utiliza una utilidad por software para habilitar el botón central como "doble click", DirectInput interpretará un click de este botón como dos clicks del botón primario.*

Para el *joystick* u otros dispositivos de juegos, *DirectInput* utiliza las calibraciones asignadas por el usuario en el *Panel de control*.

ENUMERACIÓN DE DISPOSITIVOS CON DIRECTINPUT

DirectInput es capaz, como ya se ha comentado, de obtener del sistema todos los dispositivos de entrada disponibles, determinar si están conectados y devolver información sobre ellos. Este proceso se denomina "enumeración".

Si nuestra aplicación sólo utiliza teclado estándar o ratón, o ambos, no necesitamos enumerar los dispositivos de entrada disponibles. Para otros dispositivos y sistemas con múltiples teclados o ratones, necesitaremos llamar al método *IdirectInput::EnumDevices* para obtener al menos los *GUID's* (identificadores globales) de los dispositivos que pueden ser creados.

A continuación se expone una implementación de ejemplo del método *IDirectInput::Enum Devices*:

```
GUID KeyboardGUID =
  GUID_SysKeyboard;
// LPDIRECTINPUT  lpdi;
// Esta variable se ha
// inicializado previamente
// con DirectInputCreate y
// apunta al objeto
// DirectInput

lpdi->EnumDevices(
  DIDEVTYPE_KEYBOARD,
  DIEnumDevicesProc,
  &KeyboardGUID,
  DIEDFL_ATTACHEDONLY);
```

El primer parámetro determina qué tipos de dispositivos se van a enumerar. Su valor es **NULL** si queremos enumerar todos los dispositivos sin importar del tipo que sean; en caso contrario será un valor del tipo *DIDEVTYPE_**.

Sólo necesitamos "enumerar" los dispositivos si son varios iguales

El segundo parámetro es un puntero a una función de tipo *callback* que será llamada una vez por cada dispositivo enumerado. Esta función puede crearse con el nombre que deseemos.

El tercer parámetro del método *EnumDevices* es cualquier valor de 32 *bits* que queremos pasar a la función *callback*. En este ejemplo, este valor es un puntero a una variable del tipo *GUID*, pasada a la función *callback* para que ésta pueda asignar una instancia del *GUID* del teclado.

El cuarto parámetro es un *flag* que puede requerir la enumeración de todos los dispositivos o tan sólo de aquellos que estén conectados al ordenador (*DIEDFL_ALLDEVICES* o *DIEDFL_ATTACHEDONLY*).

Si nuestra aplicación está utilizando más de un dispositivo de entrada, la función *callback* es un buen lugar para inicializar cada dispositivo que se enumere.

A continuación se presenta un ejemplo de una función *callback* que chequea la presencia de un teclado y detiene la enumeración tan pronto como encuentre este dispositivo. Seguidamente asigna el *GUID* a la instancia del último teclado encontrado mediante la variable *KeyboardGUID* (pasada como *pvRef*), que puede ser utilizada en una llamada a *IdirectInput::CreateDevice*.

```
BOOL hasEnhanced;

BOOL CALLBACK
DIEnumKbdProc(
   LPCDIDEVICEINSTANCE lpddi,
   LPVOID pvRef) {
    *(GUID*) pvRef =
     lpddi->guidInstance;
if (GET_DIDEVICE_SUBTYPE(
  lpddi->dwDevType) ==
  DIDEVTYPEKEYBOARD_PCENH) {
    hasEnhanced = TRUE;
    return DIENUM_STOP;
  }
```

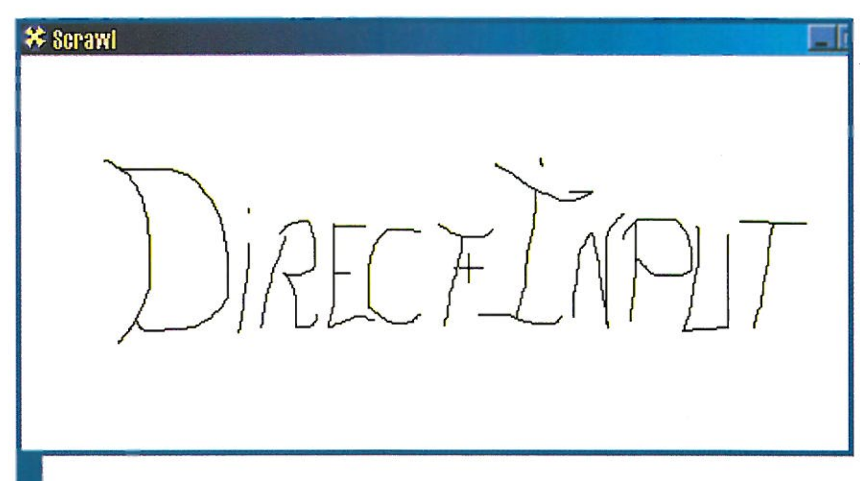


Figura 2: Ejemplo de escritura empleando *Scrawl*.



```
return DIENUM_CONTINUE;

}
// Fin de la función callback
```

El primer parámetro apunta a la estructura que contiene información sobre el dispositivo. Esta estructura es creada para nosotros de forma automática por *DirectInput*.

Con las enumeraciones podemos saber qué tipos de dispositivos hay instalados en el sistema

El segundo parámetro apunta a los datos pasados por *EnumDevices*. En este caso es un puntero a la variable *KeyboardGUID*. Esta variable fue asignada como valor por defecto anteriormente, pero tendrá un nuevo valor cada vez que se enumere un dispositivo.

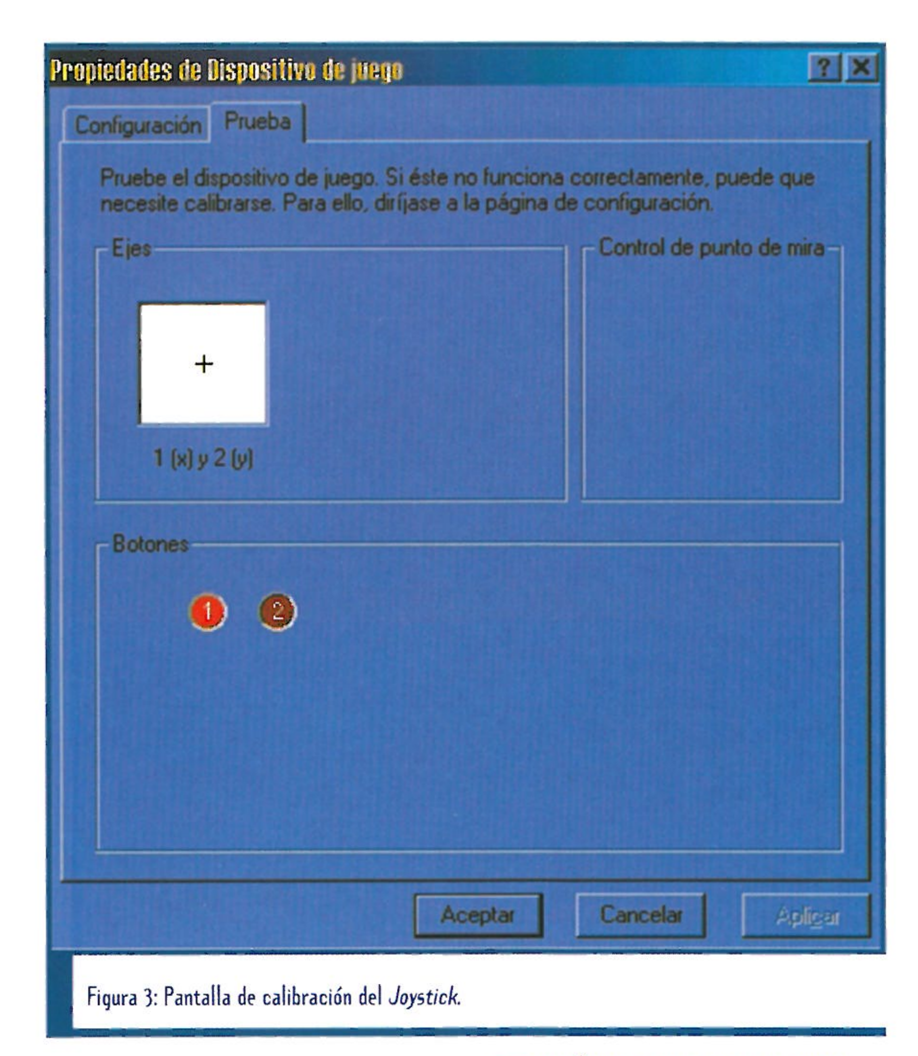


Figura 3: Pantalla de calibración del *Joystick*.



DEVICES (DISPOSITIVOS)

INICIALIZACIÓN DE DISPOSITIVOS

Nuestra aplicación debe obtener una interfaz *COM* por cada dispositivo del que se espere una entrada de datos. También debe preparar cada dispositivo para utilizarlo, por lo menos, estableciendo el formato de datos y adquiriendo el dispositivo necesario.

Las siguientes tareas detalladas son parte del propio proceso de inicialización. Siempre se requerirán ciertos pasos; otros serán necesarios si necesitamos información sobre dispositivos o si necesitamos o queremos cambiar los valores establecidos por defecto.

- Creamos el dispositivo *DirectInput* (requerido).
- Obtenemos las características del dispositivo (opcional).
- Enumeramos las teclas, botones y ejes del dispositivo (opcional).
- Establecemos el nivel cooperativo (recomendado).
- Establecemos el formato de datos (requerido).
- Establecemos las propiedades del dispositivo (opcional).
- Cuando estemos preparados para leer datos, adquirimos el dispositivo (requerido).

CREACIÓN DE UN DISPOSITIVO DIRECTINPUT

El método *CreateDevice* es utilizado para obtener un puntero a la interfaz *IdirectInputDevice*. Los métodos de ésta son posteriormente utilizados para manipular el dispositivo y obtener los datos. El siguiente ejemplo, donde *lpdi* es un puntero a la interfaz *IDirectInput*, crea un dispositivo del tipo teclado:

```
LPDIRECTINPUTDEVICE
    lpdiKeyboard;
lpdi->CreateDevice(
   GUID_SysKeyboard,
   &lpdiKeyboard, NULL);
```

El primer parámetro en *IDirectInput::CreateDevice* es un *GUID* que identifica la instancia

SÓLO PROGRAMADORES

del dispositivo para la que va a ser creada la interfaz. *DirectInput* tiene dos *GUID's* predefinidos, *GUID_SysMouse* y *GUID_SysKeyboard*, que representan el ratón del sistema y el teclado respectivamente, y podemos pasar estos identificadores dentro del método *CreateDevice*. La variable global *GUID_Joystick* no debería ser utilizada como un parámetro de *CreateDevice*, dado que es un producto *GUID*, no una instancia *GUID*.

NOTA: *Si el ordenador tiene más de un ratón, la entrada de todos ellos es combinada para formar el dispositivo de sistema. Esta misma regla se aplica para múltiples teclados.*

Para otros dispositivos diferentes del ratón y del teclado utilizaremos la instancia *GUID* para el dispositivo retornado por el método *IdirectInput::EnumDevices*. La instancia *GUID* para un dispositivo siempre será la misma. Podemos permitir al usuario seleccionar un dispositivo de una lista de todos los enumerados y guardar el *GUID* obtenido en un archivo de configuración para poder utilizarlo de nuevo en sesiones posteriores.

Si queremos utilizar los métodos de la interfaz *IdirectInputDevice2* para dispositivos *force-feedback*, debemos obtener un puntero a la interfaz de *IDirectInputDevice*.

En la siguiente función se intenta obtener la interfaz *IDirectInputDevice2* con ayuda del método *CreateDevice*. Como se puede apreciar se utilizan macros para llamar a los métodos *Release* y *CreateDevice* acorde con la sintaxis de *C* ó de *C++*.

```
HRESULT
 IDirectInput_CreateDevice2(
 LPDIRECTINPUT pdi,
 REFGUID rguid,
 LPDIRECTINPUTDEVICE2 *ppdev2,
 LPUNKNOWN punkOuter) {
   LPDIRECTINPUTDEVICE *pdev;
   HRESULT hres;

 hres =
 IDirectInput_CreateDevice(
 pdi, rguid, &pdev, unkOuter);

 if (SUCCEEDED(hres)) {
 #ifdef __cplusplus
  hres = pdev->QueryInterface(
    IID_IDirectInputDevice2,
    (LPVOID *)ppdev2);
 #else
  hres = pdev->lpVtbl->
    QueryInterface(pdev,
    &IID_IDirectInputDevice2,
    (LPVOID *)ppdev2);
 #endif
    IDirectInputDevice_Release(
  pdev);
 }
else {
  *ppdev2 = 0;
 }
 return hres;
}
```

Tabla 1. Combinaciones válidas de flags en el método IDirectInputDevice::SetCooperativeLevel

Flags	Notas
DISCL_NONEXCLUSIVE | DISCL_BACKGROUND	Valor por defecto
DISCL_NONEXCLUSIVE | DISCL_FOREGROUND	
DISCL_EXCLUSIVE | DISCL_FOREGROUND	No válido para teclado
DISCL_EXCLUSIVE | DISC_BACKGROUND	No válido para el teclado o ratón

CARACTERÍSTICAS DEL DISPOSITIVO

Antes de comenzar a preguntarnos por la entrada de un dispositivo, necesitaremos saber algo sobre sus características. ¿Tiene el *joystick* un punto de mira? ¿Está el ratón conectado al ordenador? Cuestiones de este tipo son contestadas con una llamada al método *IdirectInputDevice::GetCapabilities*, que retorna los datos en una estructura de tipo *DIDEVCAPS*. Al igual que las demás estructuras de *DirectX*, debemos inicializar el miembro *dwSize* antes de pasar esta estructura al método.

NOTA: *Para optimizar la velocidad o la memoria utilizada, podemos usar la pequeña estructura DIDEVCAPS_DX3 en lugar de la anterior.*

A continuación se presenta un ejemplo que chequea si el ratón está conectado al ordenador y si tiene un tercer eje (presumiblemente, una rueda):

```
// LPDIRECTINPUTDEVICE lpdiMouse;
// inicializado previamente

DIDEVCAPS  DIMouseCaps;
HRESULT    hr;
BOOLEAN    HasWheel;

DIMouseCaps.dwSize =
   sizeof(DIDEVCAPS);
hr =
lpdiMouse->GetCapabilities(
   &DIMouseCaps);
HasWheel = ((DIMouseCaps.dwFlags
    &
  DIDC_ATTACHED) &&
 (DIMouseCaps.dwAxes > 2));
```

Otra forma de chequear si existe otro eje o cierto botón es llamando a *IdirectInputDevice::GetObjectInfo* para ese objeto. Si la llamada devuelve *DIERR_OBJECTNOTFOUND*, el objeto no está presente. El siguiente código



determina si existe un eje Z aún cuando éste no sea el tercer eje:

```
DIDEVICEOBJECTINSTANCE didoi;
 didoi.dwSize =
sizeof(DIDEVICEOBJECTINSTANCE);
hr = lpdiMouse->GetObjectInfo(
    &didoi,DIMOFS_Z,
    DIPH_BYOFFSET);
HasWheel = SUCCEEDED(hr);
```

NIVELES COOPERATIVOS

El nivel cooperativo de un dispositivo determina cómo es compartida la entrada con otras aplicaciones y el entorno *Windows*. Esto se determina utilizando el método *IdirectInputDevice::SetCooperativeLevel*, tal y como se muestra en este ejemplo:

```
lpdiDevice->
SetCooperativeLevel(
  hwnd, DISCL_NONEXCLUSIVE |
  DISCL_FOREGROUND)
```

Los parámetros representan el *Handle* de la ventana y uno o más *flags*. Las combinaciones válidas de *flags* para este método se representan en la tabla 1:

NOTA: *Aunque DirectInput asigne valores por defecto, debemos establecer de forma explícita el nivel cooperativo, ya que es la única forma de darle a DirectInput el Handle de la ventana. Sin este Handle, DirectInput no será capaz de reaccionar a situaciones que envuelvan mensajes de Windows, tales como la recalibración del joystick. El nivel cooperativo tiene dos vertientes: cuando el dispositivo está siendo utilizado en modo foreground o background, y cuando está siendo utilizado en modo exclusivo o no exclusivo.*

FOREGROUND Y BACKGROUND

Un nivel cooperativo *foreground* significa que el dispositivo de entrada está disponible sólo cuando la aplicación está en *foreground*, es decir, tiene el foco de entrada para la ventana. Si la aplicación se mueve a modo *background*, el dispositivo será automáticamente desadquirido o no disponible.

Un nivel cooperativo *background* realmente significa *"foreground y background"*. Un dispositivo en modo *background* puede ser adquirido y utilizado por una aplicación en cualquier momento. Por norma general utilizaremos frecuentemente el modo *foreground*, puesto que a la mayoría de las aplicaciones no les interesa la entrada que tiene lugar cuando otro programa está en *foreground*.

Mientras se desarrolla una aplicación, es muy útil asignar un *DEFINE* condicional que establezca el nivel cooperativo *background* durante las operaciones de depurado del programa. Esto previene a nuestra aplicación de la pérdida de acceso al dispositivo cada vez que se mueve al *background* cuando cambiamos a la ventana de depurado.

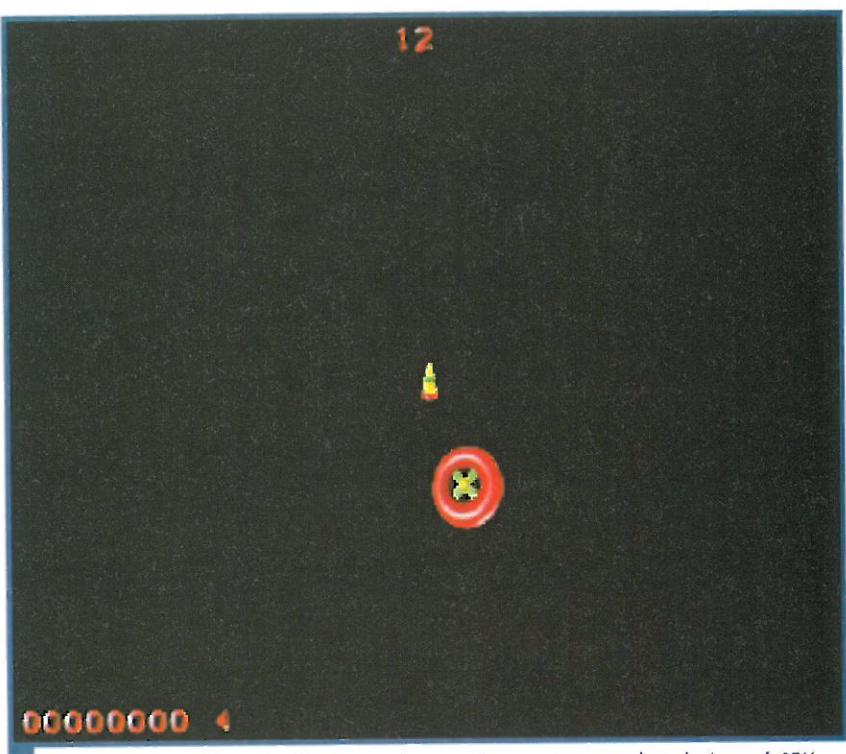


Figura 4. El uso del *ForceFeedBack* se puede comprobar con este ejemplo incluido en el *SDK*.

EXCLUSIVE Y NONEXCLUSIVE

El hecho de que nuestra aplicación esté utilizando un dispositivo a nivel exclusivo no significa que otras aplicaciones no puedan obtener datos de ese mismo dispositivo. Realmente lo que significa es que ninguna otra aplicación puede adquirir el dispositivo en modo exclusivo.

ENUMERACIÓN DEL OBJETO DEVICE (DISPOSITIVO)

Puede ser necesario para nuestra aplicación determinar qué botones o ejes están disponibles sobre un dispositivo determinado. Para saberlo debemos enumerar los

objetos del dispositivo de la misma forma que enumeramos dispositivos. *EnumObjects* solapa la funcionalidad de *GetCapabilities*. Cada método puede ser utilizado para determinar cuántos botones o ejes están disponibles. De todos modos, *EnumObjects* realmente se utiliza para catalogar todos los objetos disponibles en lugar de chequear uno en particular.

A continuación se presenta una función *callback* que simplemente extrae el nombre de cada objeto para que éste pueda ser agregado a una lista o a una matriz.

```
char   szName[MAX_PATH];

BOOL CALLBACK DIEnumDevice
     ObjectsProc(
  LPCDIDEVICEOBJECTINSTANCE
  lpddoi,LPVOID pvRef) {
  lstrcpy(szName,
     lpddoi->tszName);
  // ahora agregamos szName a
  // una lista o array(matriz)
     .
     .
     .
  return DIENUM_CONTINUE;
}
```

El primer parámetro apunta a una estructura que contiene información sobre el objeto. Esta estructura es creada para nosotros de forma automática por *DirectInput*. El segundo parámetro es un puntero a datos definido por la aplicación, equivalente al segundo parámetro de *EnumObjects*.

En este ejemplo, el parámetro no se utiliza. El valor de retorno indica que la enumeración continuará hasta que todos los objetos se hayan enumerado. A continuación podemos ver la llamada al método *EnumObjects*, que realmente pone a trabajar a la función *callback*:

```
lpdiMouse->EnumObjects(
  DIEnumDeviceObjectsProc,
  NULL, DIDFT_ALL);
```

El primer parámetro es la dirección de la función *callback*. El segundo parámetro puede ser un puntero a cualquier dato que queramos utilizar o modificar en la función. El ejemplo no utiliza este parámetro y por eso se pasa el valor **NULL**.

El tercer parámetro es un *flag* que indica qué tipo o tipos de objetos van a ser incluidos en la enumeración. En el ejemplo, todos los objetos van a ser enumerados. Para restringir la enumeración, podemos utilizar uno o más de los *flags DIDFT_**.

NOTA: *Algunos de los flags DIDFT_* son combinaciones de otros; por ejemplo, DIDFT_AXIS es equivalente a DIDFT_ABSAXIS | DIDFT_RELAXIS.*

FORMATOS DE DATOS DEL DISPOSITIVO

Establecer el formato de datos para un dispositivo es un paso esencial antes de que pueda ser adquirido y utilizado por la aplicación. El método *SetDataFormat* le dice a *DirectInput* qué objetos de dispositivo se utilizarán y cómo se gestionarán los datos.

El formato de datos es imprescindible para saber cómo interpretar los movimientos que realiza el jugador sobre los dispositivos

DirectInput tiene cuatro variables globales, *c_dfDIJoystick*, *c_dfDIJoystick2*, *c_dfDIKeyboard*, y *c_dfDIMouse*, que pueden ser pasadas en *SetDataFormat* para crear un formato de datos estándar para esos dispositivos. En este ejemplo, *lpdiMouse* es un puntero inicializado para el objeto de dispositivo "ratón":

```
lpdiMouse->SetDataFormat(
   &c_dfDIMouse);
```

NOTA: *No podemos cambiar el miembro dwFlags en la variable predefinida DIDATAFORMAT puesto que son variables constantes. Para cambiar las propiedades utilizaremos el método SetProperty después de establecer el formato de datos y antes de adquirir el dispositivo.*

PROPIEDADES DEL DISPOSITIVO

Las propiedades de los dispositivos de *DirectInput* incluyen el tamaño del *buffer* de datos, el rango de valores retornados por un eje, si el dato de los ejes es relativo o absoluto y los valores de la "zona muerta" y saturación para los ejes de un *joystick*, los cuales afectan a la relación entre la posición física de la palanca y los datos obtenidos. Los dispositivos especializados también pueden tener otras propiedades adicionales.

Salvo una excepción— la propiedad de ganancia de un dispositivo *force-feedback* — las propiedades sólo pueden ser cambiadas cuando el dispositivo está en un estado no adquirido.

Antes de llamar a los métodos *SetProperty* o *GetProperty* necesitaremos establecer una estructura de propiedades, que consiste en una estructura del tipo *DIPROPHEADER* y uno o más elementos para los datos.

Existen una gran variedad de propiedades para los dispositivos de entrada y *SetProperty* debe ser capaz de trabajar con todas



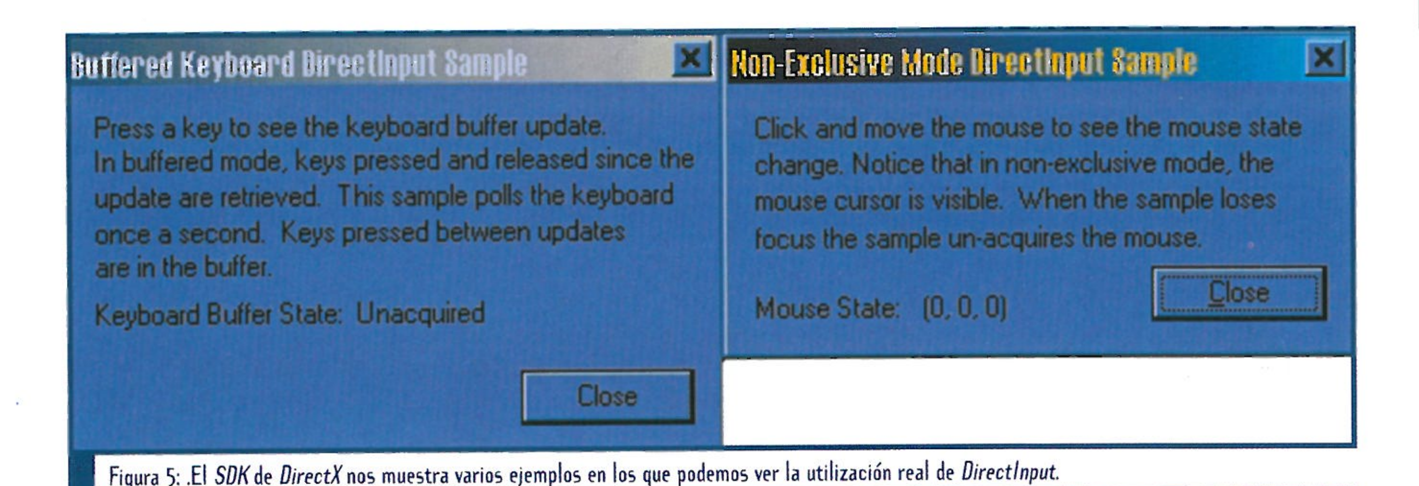


Figura 5: .El *SDK* de *DirectX* nos muestra varios ejemplos en los que podemos ver la utilización real de *DirectInput*.

ellas mediante la definición de sus propiedades. La misión de *DIPROPHEADER* es definir el tamaño de la estructura de la propiedad y cómo se interpretarán los datos. *DirectInput* incluye las siguientes estructuras de propiedad predefinidas:

- *DIPROPDWORD* define una estructura que contiene un miembro *DIPROPHEADER* y otro miembro de datos *DWORD*, para propiedades que requieran un valor simple, como un tamaño de *buffer*.
- *DIPROPRANGE* es para rango de propiedades que requieren dos valores (máximo y mínimo). Consiste en un *DIPROPHEADER* y dos miembros de datos *LONG*.

Para *SetProperty*, los miembros de datos de la estructura de propiedades son valores que queremos establecer. Mientras que para *GetProperty*, el valor actual es devuelto en esos miembros.

Antes de llamar a *GetProperty* o a *SetProperty*, la estructura *DIPROPHEADER* debe ser inicializada siguiendo uno a uno los pasos que a continuación se detallan:

- El tamaño de la estructura de la propiedad
- El tamaño de la estructura *DIPROPHEADER* propiamente dicha
- Un identificador de objeto
- Código que indica la forma en que el identificador de objeto debe ser interpretado.

DirectInput también incluye diversas estructuras de propiedad predefinidas

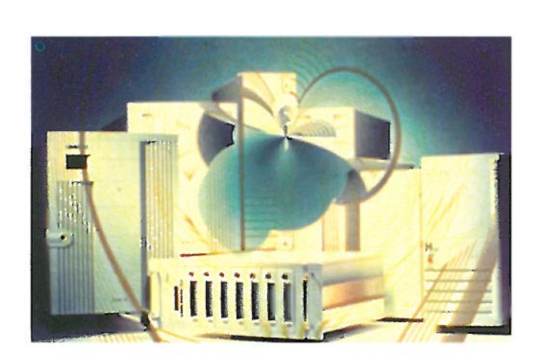
Cuando obtenemos o establecemos propiedades para un mismjo dispositivo, el identificador de objeto *dwObj* tiene un valor igual a **cero** y el código de interpretación *dwHow* es *DIPH_DEVICE*. Los siguientes valores son utilizados para identificar la propiedad pasada a *SetProperty* y *GetProperty*:

- *DIPROP_BUFFERSIZE*
- *DIPROP_AXISMODE*
- *DIPROP_CALIBRATIONMODE*
- *DIPROP_GRANULARITY*
- *DIPROP_FFGAIN*
- *DIPROP_FFLOAD*
- *DIPROP_AUTOCENTER*
- *DIPROP_RANGE*
- *DIPROP_DEADZONE*
- *DIPROP_SATURATION*

El siguiente ejemplo establece el tamaño del *buffer* para un dispositivo que albergará hasta diez entradas de datos:

```
DIPROPDWORD dipdw;
HRESULT hres;
dipdw.diph.dwSize =
  sizeof(DIPROPDWORD);
dipdw.diph.dwHeaderSize =
  sizeof(DIPROPHEADER);
dipdw.diph.dwObj = 0;
dipdw.diph.dwHow =
  DIPH_DEVICE;
dipdw.dwData = 10;
hres =
lpdiDevice->SetProperty(
  DIPROP_BUFFERSIZE,
  &dipdw.diph);
```

Con esto finalizamos el repaso a los componentes más importantes de *DirectX* que hemos estado realizando. Ahora sólo nos queda esperar y ver qué mejoras (al parecer muchas) nos deparará la nueva versión, que dentro de poco estará en los servidores de *Internet* a nuestra disposición. Pero si queréis más información podéis dirigiros directamente a la página *web* que *Microsoft* ha creado para todo lo que está relacionado con *DirectX*: **www.microsoft.com/directx/default.asp**



Desarrollo cliente/servidor (III)

Javier Toledo (jtoledo@sei.es)

Si en el artículo anterior enfocamos el sistema cliente/servidor desde el punto de vista del cliente, ahora le toca el turno al servidor. En esta entrega exponemos cuáles son las características, el proceso de diseño y creación de una base de datos, tanto en *SQL Server* como en *Oracle*.

¿QUÉ ES UNA BASE DE DATOS?

La funcionalidad de una base de datos es almacenar un conjunto de datos organizados de tal manera que sea fácil consultarlos y modificarlos. Dentro de ellas se construyen ubicaciones específicas, es decir, tablas para cada tipo de datos que se va a almacenar. Por tanto, hablamos de bases de datos bien diseñadas al referirnos a las que poseen un conjunto de tablas bien organizado y estructurado.

Un servidor de bases de datos es un *software* que se ocupa de su mantenimiento y facilita el acceso. Estos servidores administran y organizan las bases de datos a un nivel físico y a un nivel lógico. A nivel físico cada servidor de bases de datos almacena la información en un conjunto de ficheros gestionados donde residen los datos. Por lo tanto, salvo en el proceso de creación de una base de datos, el usuario se desentiende de donde se almacenará la información físicamente; esta tarea corresponde al servidor. Sin embargo, el usuario podrá disponer de la información almacenada valiéndose de su conocimiento del diseño lógico.

Una BBDD está bien diseñada si sus tablas están bien organizadas

Supongamos que un usuario se conecta a nuestra base de datos con intención de obtener la relación de los distintos comercios con los que trabajamos. Sólo necesitará saber que la información se encuentra almacenada en la tabla COMERCIOS y solicitarla utilizando un lenguaje que entienda nuestro servidor, el *SQL*.

BASES DE DATOS RELACIONALES

El modelo relacional es el sistema de organización lógico de la información más utilizado actualmente. Fue desarrollado por E.F. Codd a finales de los años sesenta. Las bases de datos relacionales, como lo son *SQL Server* y *Oracle,* organizan su información siguiendo este esquema. La información es almacenada en tablas compuestas de filas y columnas donde a cada columna se le asocia un tipo de datos, por ejemplo numérico o de texto. Las filas de cada tabla contienen los registros. Cada fila de la tabla *COMERCIOS* representa los datos que tenemos disponibles de un determinado comercio.

Los registros de una tabla se identifican mediante las claves. Una clave es una columna o un con-



junto de columnas que identifican singularmente a una fila de la tabla. La clave de nuestra tabla *COMERCIOS* es el campo *ID*. De este modo si estamos en posesión del *ID* de un determinado comercio podremos modificar sus datos de la siguiente manera

```
UPDATE COMERCIOS SET
     COMERCIOS.DIRECCION = "DIR2"
WHERE COMERCIOS.ID=1
```

Es decir, si estamos en posesión de la clave de un registro de nuestra tabla, podremos operar con los datos de este registro utilizando su campo clave en nuestras operaciones. No existirán dos comercios con el mismo *ID* si en la creación de nuestra tabla se especificó el campo *ID* como campo clave.

NORMALIZACIÓN

Cuando diseñemos el esquema de nuestra base de datos, es decir, su estructura lógica, se nos plantearán diversas alternativas para organizar nuestros datos. ¿Es mejor utilizar una tabla grande donde almacenemos la mayoría de nuestros datos o es mejor la alternativa de crear muchas tablas más pequeñas?. *Codd* proporcionó un proceso denominado normalización de las bases de datos, que consiste en optimizar la forma de almacenar los datos en las tablas. Siguiendo dicho proceso se establecerá un sistema de mantenimiento de la consistencia de la información y se minimizará el número de columnas de nuestras tablas con el fin de eliminar información repetitiva.

Por ejemplo, en nuestra sencilla base de datos tenemos que almacenar los distintos productos que tenemos disponibles en cada uno de nuestros comercios y qué cantidad tenemos de los mismos. Una manera de almacenar esta información utilizando una única tabla podría haber sido la Tabla de Comercios 1.

Supongamos que tenemos un comercio que distribuya dos tipos de componentes *hardware* **discos duros** y **discos flexibles de 3.5**.Con el diseño de tabla anterior podríamos almacenar sin ningún problema dichos datos. Necesitaríamos insertar dos registros en nuestra tabla. El primero de ellos estaría compuesto por el nombre, la dirección, ciudad y teléfono de nuestro comercio, seguido del nombre del producto **discos duros** y del número de unidades disponibles. El segundo registro de nuestra tabla sería idéntico en lo que a los datos del comercio se refiere, pero variaría en el nombre y unidades del producto. Salvo los datos de los dos últimos registros, el resto estarán duplicados un número de veces igual a la cantidad de productos diferentes.

El proceso de normalización de la base de datos nos hubiese establecido un diseño distinto para evitar esto, basado tres tablas que serían Tabla de Comercios 2, Tabla de Productos y Tabla de Comercios-Productos.

Para nuestro ejemplo anterior, en la tabla *Comercios* tendríamos un solo registro indicando los datos de nuestro comercio. En la tabla productos tendríamos los datos de los productos **discos duros** y **discos flexibles de 3.5**. Con la tabla *Comercios-Productos* podríamos saber cuáles son los productos disponibles en cada comercio y el número de los mismos.

El proceso de normalización se divide en cinco niveles. En la mayoría de los casos no interesa llegar a normalizar una base de

Tabla de Comercios 1

Nombre del campo	Tipos de datos
IDENTIFICADOR	Numérico (campo clave)
NOMBRE_COMERCIO	Texto
DIRECCIÓN_COMERCIO	Texto
CIUDAD	Texto
TELÉFONO	Texto
NOMBRE_PRODUCTO	Texto
UNIDADES	Numérico

Tabla de Comercios 2

Nombre del campo	Tipos de datos
IDENTIFICADOR_COMERCIO	Numérico (campo clave)
NOMBRE_COMERCIO	Texto
DIRECCIÓN_COMERCIO	Texto
CIUDAD	Texto
TELÉFONO	Texto

Tabla de Productos

Nombre del campo	Tipos de datos
IDENTIFICADOR_PRODUCTO	Numérico (campo clave)
NOMBRE_PRODUCTO	Texto

Tabla de Productos-Comercios

Nombre del campo	Tipos de datos
IDENTIFICADOR_COMERCIO	Numérico
IDENTIFICADOR_PRODUCTO	Numérico
CANTIDADES	Numérico

datos mas allá del 3º o 4º nivel ya que el proceso de normalización disminuye la redundancia de la información dispersando los datos en distintas tablas. Esto tiene su coste en el rendimiento de nuestras aplicaciones, ya que las consultas serán más dispersas (utilizarán más tablas) y su tiempo de procesamiento será mayor.

ESTRUCTURA LÓGICA Y FÍSICA

Para el entendimiento y comprensión del funcionamiento y la estructura de una base de datos disponemos de dos puntos de vista, el lógico y el físico. La arquitectura física de la base de datos hace referencia al conjunto de ficheros que la componen. Sin embargo, la estructura lógica está compuesta por los objetos que nosotros manejaremos al utilizar dicha base. Es decir, en nuestra base de datos de comercios, nosotros sabremos que toda la información se almacena en ficheros del disco duro de nuestro servidor. Sin embargo, manejaremos la información utilizando las tablas de la base de datos sin tener conocimiento de dónde se están almacenando los datos físicamente.

En una base de datos lo importante es su estructura lógica, no la física

Oracle es un sistema de gestión de bases de datos relacionales. (*RDBMS Relational Database Management System*). Su arquitectura física incluye un número de ficheros en disco entre los que podemos destacar tres tipos:

- Archivos de datos.
- Ficheros de control.
- Ficheros de *redo log*.



Figura 1. Asistente de bases de datos Oracle - Ficheros de Redo Log.

ARCHIVOS DE DATOS

Son los archivos donde se almacenan los datos de nuestras tablas y de nuestros usuarios. *Oracle* se compone de uno o varios de estos archivos que se pueden agrupar para formar *tablespaces* (espacios de tablas). *Oracle* crea estos archivos de datos utilizando bloques para manejar los datos almacenados.

FICHEROS DE CONTROL

Son ficheros de pequeño tamaño pero que manejan toda la información de la base de datos. Dentro de los ficheros de control se almacena información crucial, como por ejemplo, todos los nombres de los ficheros de datos, el nombre de la propia base de datos, información para la sincronización, etc. Toda esta información es actualizada constantemente. De hecho estos archivos suelen duplicarse en distintos discos ya que su contenido es crucial y la perdida o deterioro de la información almacenada representaría un serio problema.

FICHEROS DE REDO LOG.

Como mínimo debe haber dos ficheros de este tipo para cada base de datos. Aquí se registran los cambios realizados. De tal manera que ante una operación que cambie el contenido de la base de datos, se almacenarán en estos ficheros tanto los valores antiguos como los nuevos.

Es decir, ante una instrucción del tipo *Update*, en los ficheros de *redo log* se almacenarán los datos de los registros modificados por el *Update* antes y después de dicha modificación. Cuando se producen transacciones en la base de datos, se introducen los cambios en los *buffers* del registro de rehacer, mientras que los bloques de datos afectados por la transacción no se escriben de manera inmediata en el disco, sino por bloques. De esta forma se optimiza el rendimiento de las escrituras a los ficheros de datos.

Supongamos que una transacción finaliza satisfactoriamente, pero en el mismo momento en que el usuario la valida nuestro servidor de base de datos sufre una caída. La transacción ha sido validada, pero al servidor de base de datos no le ha dado tiempo a escribir en los archivos de datos la información correcta. Gracias a los ficheros de *redo log* podemos trasladar la operación a nuestros archivos de datos, ya que en ellos sí está almacenada toda la información necesaria para completar la transacción.

REGISTROS DE REDO LOG ARCHIVADOS

Oracle escribe en los archivos de rehacer en línea de manera cíclica, es decir, después de llenar el primer archivo de registro escribe



en el segundo, hasta rellenarlo. Cuando se han llenado todos los archivos de registro de rehacer en línea *Oracle* vuelve sobre el primero y empieza a sobreescribir su contenido con nuevos datos de transacciones. Sin embargo, podemos configurar nuestra base de datos para que se realicen copias de los archivos de registro de rehacer en línea antes de sobreescribirlos. Estas copias pueden utilizarse para recuperar parte de la base de datos al estado que tuviese en cualquier instante. Estas copias denominadas registros de *redo log archivados* forman parte fundamental de la mayoría de los *backup* de nuestra base datos.

ESTRUCTURA LÓGICA DE UNA BASE DE DATOS ORACLE

Es la estructura que realmente manejamos, pues normalmente nos desentendemos del aspecto físico del almacenamiento de datos.

TABLESPACES

Los *tablespaces* o espacios de tablas son una división lógica. Cada base de datos debe tener al menos uno de estos espacios, ya que si no, no existiría almacenamiento físico para nuestros datos. Cada *tablespace* consiste en uno o más ficheros del sistema operativo que puede ser creado y activado mientras la base de datos está en línea. Puede ser puesto fuera de línea, excepto para el *tablespaces system*, o sobre un *tablespace* con segmentos de *rollback* en línea.

El *tablespace system* es un *tablespace* obligatorio para todas las bases de datos *Oracle*. Es necesario para las operaciones internas. Contiene información del diccionario de datos donde se almacenan los procedimientos guardados, los paquetes y los *triggers*. El resto de los *tablespaces* contendrán los distintos objetos de las aplicaciones, como las tablas y los índices.

SEGMENTOS

Son objetos donde se almacenan los datos. Representan el almacenamiento de los objetos lógicos que manejan los usuarios, como las tablas, índices, etc. Existen distintos tipos de segmentos:

- Datos.
- Índices.
- *Rollback*.
- Temporales.
- Caché.

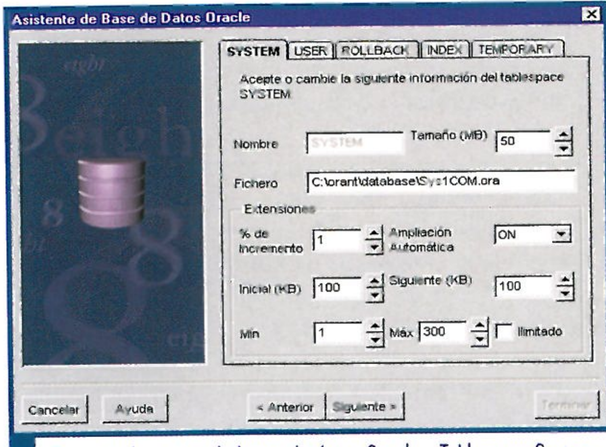


Figura 2. Asistente de bases de datos Oracle – Tablespace System

En los segmentos de datos se almacenan las tablas de los usuarios. Los segmentos temporales son creados automáticamente por *Oracle* para realizar operaciones complejas como las ordenaciones, los *group by* y los *joins*. Estos segmentos se crean automáticamente en los *tablespaces* cuando no hay espacio suficiente en la memoria. Los segmentos de caché no son accesibles para los usuarios. Se crean en el *tablespace system* y almacenan información sobre la definición de ciertas tablas del diccionario de datos para un mejor funcionamiento del servidor.

Los segmentos de *rollback* representan el espacio físico del que *Oracle* se valdrá para almacenar los diferentes estados de la información de nuestra base de datos debido a las diferentes transacciones de los usuarios. Supongamos que dos diferentes usuarios están realizando transacciones sobre nuestra tabla COMERCIOS. Los dos usuarios se conectan a la base datos en el mismo instante de tiempo cuando existen cinco comercios en nuestra tabla. El usuario *A* añade un nuevo comercio. En ese momento, y antes de que el usuario *A* realice un *commit* sobre esta transacción, (validar los cambios que ha realizado) para el usuario *B* sólo existen cinco comercios en la tabla. Esto es así, debido a que el usuario *A* todavía no ha validado su transacción. Por lo tanto *Oracle* necesita disponer de los medios necesarios para presentar distintos contenidos de la tabla de comercios según sea el caso.

LAS TABLAS

Son en el mecanismo de almacenamiento de los datos. Constan de un conjunto fijo de columnas que describen los atributos de los datos almacenados. Cada columna tiene un nombre y unas características específicas.

PROCESO DE CREACIÓN DE UNA BASE DE DATOS ORACLE

A la hora de crear una nueva base de datos *Oracle* podemos utilizar dos métodos. Bien con el uso de un asistente, o bien lanzando contra la base datos todas las instrucciones que la crearán en el servidor. A continuación presentamos cuáles son estas instrucciones.

```
set ORACLE_SID=COM

C:\orant\bin\oradim80 -new -sid
     COM -intpwd pwdinterna -
     startmode auto -pfile
     C:\orant\database\initCOM.ora

C:\orant\bin\oradim80-startup -sid
     COM -starttype srvc,inst -
     usrpwd pwdinterna -pfile
     c:\orant\database\initCOM.ora

C:\orant\bin\svrmgr30
```

Para ejecutar estas instrucciones lo primero que debemos hacer es abrir una ventana de *MS-DOS* e ir ejecutándolas una a una, o bien ejecutar el fichero **SQLCOM.BAT** que adjuntamos con el *CD-ROM*.

La primera instrucción crea una variable de entorno llamada *ORACLE_SID*, que debe contener el nombre de la nueva base de datos. El programa **oradim80** creará los servicios de *Windows NT* encargados de arrancar y parar muestra base de datos. A continuación utilizaremos la herramienta **svrmgr30**, herramienta de administración de *Oracle* en modo de comandos, desde la que crearemos nuestra base datos.

```
C:\orant\bin\svrmgr30
```

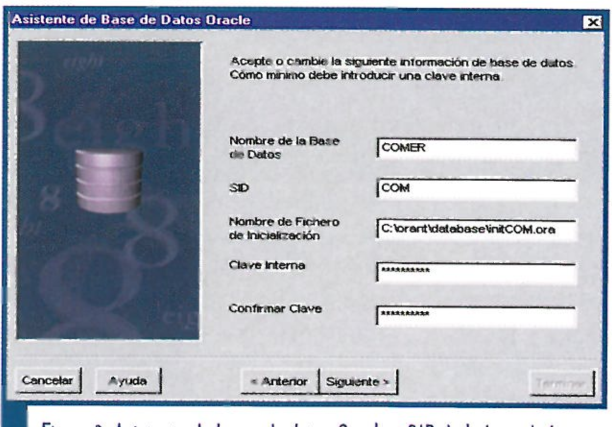


Figura 3. Asistente de bases de datos Oracle - SID de la base de base de datos y contraseña de internal.

```
spool C:\orant\database\spoolmain
set echo on
connect INTERNAL/pwdinterna
startup nomount
     pfile=C:\orant\database\
     initCOM.ora
CREATE DATABASE COMER
CONTROLFILE REUSE
LOGFILE 'C:\orant\database\
     logCOM1.ora' SIZE 1024K,

     'C:\orant\database\logCOM2.cr
     a' SIZE 1024K
MAXLOGFILES 32
MAXLOGMEMBERS 2
MAXLOGHISTORY 1
DATAFILE
     'C:\orant\database\Sys1COM.or
     a' SIZE 50M
MAXDATAFILES 254
MAXINSTANCES 1
CHARACTER SET WE8ISO8859P1
NATIONAL CHARACTER SET
     WE8ISO8859P1;
spool off
```

La instrucción *spool* provoca que se cree el fichero *C:\orant\database\spoolmain* donde se almacenarán todos los mensajes generados como resultado de las instrucciones que vamos a lanzar a continuación. Con la instrucción *connect* nos conectaremos como el usuario *INTERNAL* que tiene los permisos necesarios para crear nuestra base de datos.

Debemos asegurarnos de la existencia del fichero *initCOM.ora* en el directorio *C:\orant\database* antes de ejecutar las siguientes sentencias.

La instrucción *CREATE DATABASE ***NOMBRE BASE DE DATOS *** creará nuestra base de datos. Dicha instrucción lleva una serie de parámetros que especifican sus características. Por ejemplo, el parámetro *WE8ISO8859P1* especifica el juego de caracteres que contendrá, el parámetro *LOGFILE* indicará cuáles van a ser nuestros ficheros de rehacer en línea y el parámetro *DATAFILE* especifica cuál va a ser el fichero de datos que contendrá el *tablespace system*. (Recordemos que todas las bases de datos *Oracle* necesitan un *tablespace system*).

De esta forma ya tenemos creada nuestra base de datos. En la próxima entrega crearemos un usuario y las tablas de nuestra

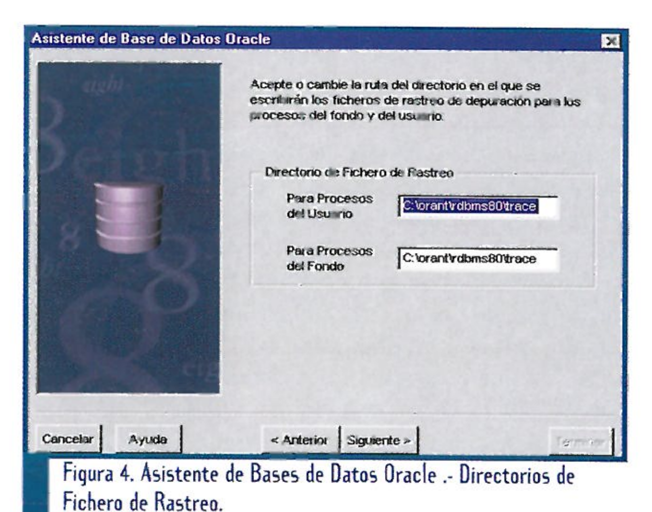


Figura 4. Asistente de Bases de Datos Oracle .- Directorios de Fichero de Rastreo.



Listado1. Creación de los tablespaces de la base de datos

```
spool C:\orant\database\spoolmain
set echo on
connect INTERNAL/pwdinterna
ALTER DATABASE DATAFILE
     'C:\orant\database\Sys1COM.o
     ra' AUTOEXTEND ON;
CREATE ROLLBACK SEGMENT SYSROL
     TABLESPACE "SYSTEM"
     STORAGE (INITIAL 100K NEXT
     100K);
 ALTER ROLLBACK SEGMENT "SYSROL"
     ONLINE;

**  LANZAMOS LOS SCRITPS DE
     CREACIÓN DEL CATÁLOGO **

@C:\orant\Rdbms80\admin\
     catalog.sql;
@C:\orant\Rdbms80\admin\
     catproc.sql
@C:\orant\Rdbms80\admin\caths.sql

REM ****TABLESPACE DE ROLLBACK***

CREATE TABLESPACE RBS DATAFILE
     'C:\orant\database\
     Rbs1COM.ora' SIZE 10M
DEFAULT STORAGE ( INITIAL 1024K
     NEXT 1024K MINEXTENTS 2

     MAXEXTENTS 121 PCTINCREASE
     0);
ALTER DATABASE DATAFILE
     'C:\orant\database\Rbs1COM.o
     ra' AUTOEXTEND ON;

REM **ALTERA LOS PARÁMETROS DE
     ALMACENAMIENTO DEL TABLESPA-
     CES SYSTEM **

ALTER TABLESPACE SYSTEM
DEFAULT STORAGE ( INITIAL 100K
     NEXT 100K MINEXTENTS 1
     MAXEXTENTS 300 PCTINCREASE
     1);

REM ****TABLESPACE DE USUARIO ***
CREATE TABLESPACE TABLESPACECO-
     MERCIO
     DATAFILE 'C:\orant\
     database\Usr1COM.ora' SIZE
     3M
     DEFAULT STORAGE ( INITIAL
     50K NEXT 50K MINEXTENTS 1
     MAXEXTENTS 121 PCTINCREASE
     1);
ALTER DATABASE DATAFILE
     'C:\orant\database\
     Usr1COM.ora' AUTOEXTEND ON;

REM *****TABLESPACE TEMPORAL*****
CREATE TABLESPACE TEMPORARY
     DATAFILE 'C:\orant\
     database\Tmp1COM.ora' SIZE
     10M
     DEFAULT STORAGE ( INITIAL
     100K NEXT 100K MINEXTENTS 1
     MAXEXTENTS 121 PCTINCREASE
     0) TEMPORARY;
ALTER DATABASE DATAFILE
     'C:\orant\database\Tmp1COM.o
     ra' AUTOEXTEND ON;

REM ** CREACIÓN DE DOS SEGMENTOS
     DE ROLLBACK ***

CREATE PUBLIC ROLLBACK SEGMENT
     RB0 TABLESPACE "RBS"
STORAGE ( INITIAL 50K NEXT 50K
     MINEXTENTS 2 MAXEXTENTS 121
     );

CREATE PUBLIC ROLLBACK SEGMENT
     RB1 TABLESPACE "RBS"
STORAGE ( INITIAL 50K NEXT 50K
     MINEXTENTS 2 MAXEXTENTS 121
     );

ALTER ROLLBACK SEGMENT "RB0"
     ONLINE;
ALTER ROLLBACK SEGMENT "RB1"
     ONLINE;
alter user sys temporary
     tablespace TEMPORARY;
alter user system default
     tablespace TABLESPACECOMER-
     CIO;
alter rollback segment "SYSROL"
     offline;
spool off
```

aplicación. Una vez creado nos conectaremos a la base de datos asumiendo su identidad. De este modo, cuando creemos las tablas necesarias para nuestra aplicación, dichas tablas pertenecerán a este nuevo usuario.

Los objetos de este usuario necesitarán ser creados en un *tablespace*. Nuestra base de datos posee un *tablespace system*, pero no es el indicado para almacenar dichos objetos, ya que está destinado a propósitos de gestión de nuestra base datos. Aunque no sea necesario, si es aconsejable que antes de crear a nuestro usuario creemos los *tablespaces* adecuados. Por lo tanto crearemos un *tablespace RBS* destinado a almacenar los segmentos de *rollback*. El *tablespace TABLESPACECOMERCIO* será el que contenga las tablas de dicho usuario y *TEMPORARY* será nuestro *tablespace* temporal.

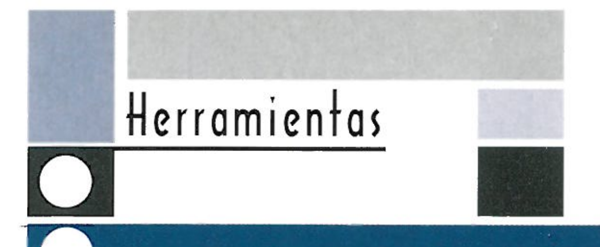
IMPLEMENTACIÓN BÁSICA DE LA BASE DE DATOS ORACLE

En su forma más elemental, una sede datos *Oracle* consta de:

- Uno o más archivos de datos.
- Uno o más archivos de control.
- Dos o más registros de rehacer en línea (*redo log*).

Como estructura interna contendrá:

- Varios usuarios.
- Uno o más segmentos de *rollback*.
- Uno o más espacios de tablas (*tablespaces*).
- Tablas del diccionario de datos.
- Objetos de usuarios (tablas, índices, de vistas, etc..).





Solaris 7. Instalación y primer contacto

Vicente Antonio Sánchez Werner (Mclead@ssc.es)

Solaris es el Sistema Operativo de la empresa *Sun Microsystems*, que en su versión 7 para las plataformas x86 y *Sparc* pretende llegar a un número de usuarios más amplio que en el caso de sus versiones anteriores.

INTRODUCCIÓN

Aunque apareció a finales del año pasado, es recientemente cuando este sistema operativo (*S.O.*) está cogiendo impulso. Esto se debe principalmente a la iniciativa de *Sun* de ofrecerlo a bajo coste para su uso no comercial. Con la nueva versión de este sistema *Unix* pretende llegar a un amplio espectro de desarrolladores potenciales que aporten nuevos programas e incrementen su base de usuarios.

DESCRIPCIÓN

La versión que hemos tenido ocasión de probar en una máquina *x86* que cumplía todas las especificaciones, es la que *Sun* proporciona a bajo coste con su programa *Developers Connection*. *Solaris 7* está disponible en varias versiones más dependiendo de su uso. También existen paquetes de *software* complementario que podemos adquirir de la compañía para añadirle más funcionalidad.

Este *S.O.* viene presentado en una pequeña carpeta donde encontramos 3 *CD-ROM's*, un disquete y diversos documentos: la licencia de uso, una guía de inicio.... Los tres *CD-ROM's* son *Solaris* para *x86*, para *Sparc* y la documentación en formato *HTML*; el disquete lleva la etiqueta *Device configuration assistant* y es el instrumento que permitirá la instalación del *S.O.* en nuestro ordenador.

La guía de iniciación, en seis idiomas, indica los requisitos mínimos del *S.O*, dónde encontrar la documentación necesaria para instalarlo, métodos posibles de instalación y una introducción al procedimiento de la misma.

El *CD-ROM* de documentación se inicia con una serie de documentos sobre *Solaris 7* en todos los idiomas. La instalación en formato *HTML* está en inglés , pero el procedimiento de instalación de *Answer Book2* lo está en todos. Si queremos leer los documentos en nuestro *PC* sin entrar en *Solaris* o en *Unix*, los *links* entre páginas de diferentes directorios no funcionarán adecuadamente.

Solaris está disponible en las plataformas x86 y Sparc

Los requisitos técnicos del sistema son 32 *Mb* de *RAM* y 700 *Mb* de disco duro, pero lo fundamental es que nuestro *hardware* esté en la lista de compatibilidad de *Solaris*, que se puede encontrar en el *CD-ROM* dentro del directorio *\Solaris_\common\html_en\hardware\book.htm* de nuestra unidad. Se puede instalar *Solaris* reducido en equipos de menores prestaciones, sacrificando funcionalidades.

NOVEDADES

Una de las más importantes mejoras de esta versión es la incorporación de un *kernel* nativo de 64 *bits* para las estaciones de trabajo que incorporen procesadores *UltraSparc,* que mejora mucho la velocidad respecto al anterior de 32 *bits*.

Solaris utiliza como sistema de archivos *UFS*, al que se han añadido nuevas funcionalidades y mejoras, como la opción de *logging*, que reduce de manera notable el tiempo de arranque del *S.O.* tras un fallo y elimina la necesidad de ejecutar el comando *fsck* periódicamente. La posibilidad de ignorar las actualizaciones de las fechas y horas de modificación de los archivos en determinados sistemas, reduce la actividad de los discos (especialmente a los que se accede frecuentemente).

Se ha mejorado el servicio de correo al actualizar su agente al programa *sendmail 8.9* que incorpora novedades interesantes, como la posibilidad de filtrar el *spam mail* y una nueva sintaxis que simplifica el proceso de configuración. También se han añadido dos nuevos comandos: *pkill* y *pgrep* que permiten la eliminación y visualización, de los procesos que haya en el sistema cuyos atributos coincidan con los suministrados a estos comandos.

El uso de UFS Logging reduce el tiempo de arranque del servidor tras un fallo del sistema

Una novedad de especial interés es la posibilidad de configurar el volcado del sistema en caso de error (*system crash dump*), que aparece cuando existe un fallo grave en la ejecución de una aplicación. Ahora los datos son volcados en formato comprimido -con el consiguiente ahorro de espacio en disco- y es posible configurar cómo y dónde se vuelcan estos fallos, e incluso hacer que suceda en un segundo plano si hemos definido un periférico dedicado como receptor de estos volcados. Esto supone una mayor velocidad de operación y se facilita enormemente el rastreo de los fallos del sistema.

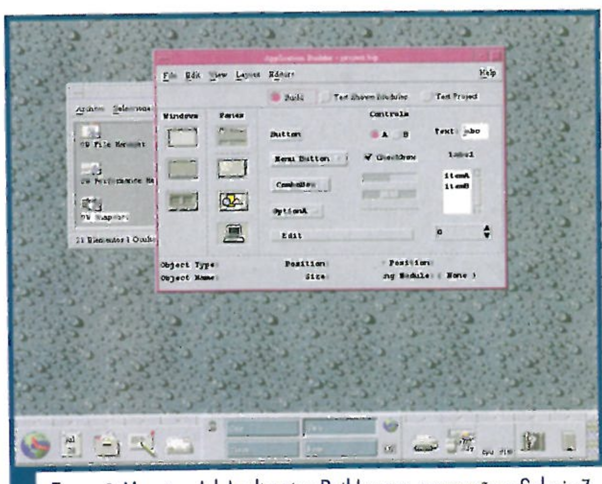


Figura 1. Muestra del Application Builder que acompaña a Solaris 7.

En el ámbito de los estándares, *Solaris 7* ha recibido la certificación del *Unix 98* e incorpora *Motif 2.1* y *CDE* como escritorio para una más sencilla operación y administración del sistema.

INSTALACIÓN

La instalación de *Solaris 7* en nuestro ordenador no es tan sencilla como cabría suponer, teniendo en cuenta la competencia a la que se enfrenta en la plataforma *x86*. Lo primero es comprobar si todo nuestro *hardware* es compatible con *Solaris* y/o tenemos *drivers* para este *S.O.*; debemos prestar atención a las tarjetas de red, vídeo, sonido, unidades de *backup* y todos los periféricos que no sean habituales.

Las tarjetas de vídeo y de sonido pueden ser prescindibles ya que podemos optar por no utilizar sus servicios o hacerlo al mínimo (modo *vga* 640x480 y 16 colores). Los periféricos para los que no tengamos *drivers* o no estén soportados por *Solaris* no serán utilizables.

Tras este paso deberemos hacer un *backup* completo de los datos que poseamos en el disco duro, ya que *Solaris* toma el control completo del disco eliminando los datos existentes.

El procedimiento para instalar *Solaris* consiste en arrancar el disquete de instalación y colocar el *CD-ROM* de *Solaris 7* para *x86* en el *CD-ROM*. El disquete arranca y va dirigiendo al usuario: primero se hace un reconocimiento del *hardware* del sistema y muestra los resultados, por si queremos modificar la lista del *hardware* detectado, - si disponemos de teclado en castellano es bueno cambiar su configuración, ya que por defecto asume el inglés -. Una vez superada esta fase, da a elegir el dispositivo desde el que continuaremos la instalación. En este punto se propone el *CD-ROM* y el disco duro. En este caso elegiremos el primero y continuaremos.

El sistema arrancará el *kernel* desde el *CD-ROM* y pedirá que se elija entre tres opciones de instalación, que tienen sus pros y sus contras - explicados por encima en la guía de instalación - pero si nuestro ordenador cumple con los requisitos de la instalación *Web Start*, ésta es la opción más sencilla de todas al utilizar una interfaz gráfica.

La instalación *Jump Start* no está documentada en la guía impresa ni entre la documentación del *CD-ROM*. La instalación interactiva es aconsejable en determinados casos y es necesaria cuando nuestra máquina no cumple los requisitos de una instalación *Web Start*. Este último tipo de instalación es aconsejable cuando nuestro ordenador no tiene ningún dato en el disco duro o sea muy personalizada. En el primero de estos casos, selecciona automáticamente los parámetros del disco duro y lo particiona sin nuestra intervención.

Sin embargo, la instalación *Web Start* no realiza esta función y nos obliga a efectuarla manualmente. En el otro caso, esta instalación permite un control mucho mayor sobre los paquetes de *software* que se instalan y cuáles no, lo que permite ahorrar espacio. Finalmente si nuestra máquina no posee al menos 64 *Mb* de *RAM* o no cumple los requisitos de disco de *Web Start*, este método de instalación sí que nos permitirá instalar el sistema.

NOTA: En algún momento el programa intenta configurar el entorno gráfico, pero no siempre lo consigue. Si hacemos caso de las instrucciones conseguiremos que el ordenador se bloquee porque no arrancarán las X-Windows. Si confiuramos manualmente el entorno gráfico se realizará correctamente.

Para la redacción de este artículo se efectuó una instalación *Web Start* para evaluar sus capacidades. Éste nos guía a lo largo de la instalción, indicándonos cómo arrancar el *kernel* y configurar dispositivos. Después se elige el idioma.

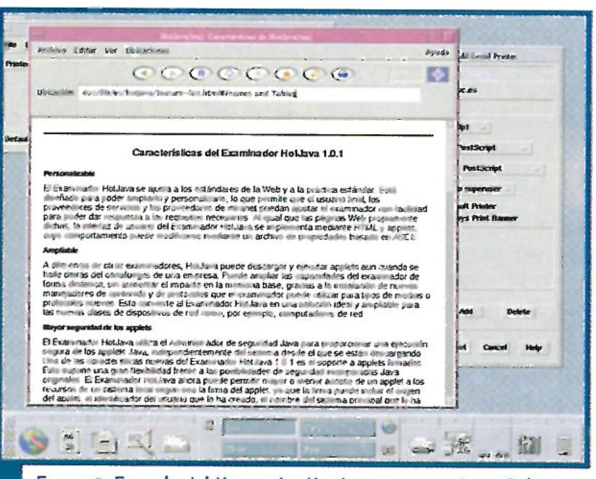


Figura 2. Ejemplo del Navegador HotJava incorporado en Solaris.

Si nuestro disco duro no contiene una partición *Solaris*, el procedimiento de instalación se aborta y nos remite a una instalación interactiva, para salir a un *shell* del sistema desde el que podremos efectuar un *format* y un *fdisk*, de forma que el programa reconozca la presencia del disco duro y acceda al mismo. Podemos obviar este paso si utilizamos una instalación interactiva que configura de forma autónoma el disco duro. De todas formas, si poseemos los parámetros físicos del disco duro podemos salir en la primera pantalla que aparezca tras aceptar la instalación interactiva y en el *shell* del sistema introducir la siguiente orden:

```
#format
```

El comando nos preguntará los parámetros físicos del disco duro y después nos dejará en el *prompt* del programa. El paso siguiente consiste en escribir *fdisk* y crear una partición que abarque todo el disco duro o el porcentaje que nos interese, tras lo cual saldremos, reiniciaremos el sistema y volveremos a efectuar el procedimiento de instalación hasta llegar a este punto.

Si tenemos ya definida la partición de *Solaris* y configurado el entorno gráfico, arrancarán las *X-Window* y se iniciará la última parte de la instalación. En este proceso tendremos que indicar un nombre para el ordenador y la zona horaria en la que nos encontramos; tras lo cual arrancará propiamente el sistema *Web Start* de instalación.

Este sistema está basado en *Java* y funciona a través del navegador *Hot Java*; tiene una interfaz gráfica agradable y sencilla de utilizar, así como unas instrucciones de instalación que nos guían a cada paso.

El proceso permite efectuar una instalación según cuatro perfiles predefinidos, así como decidir qué *software* opcional se va a instalar junto al sistema. Una vez seleccionado el perfil o la instalación por defecto, el programa pregunta:

1) Si deseamos asignar espacio para tener el *software* de arranque de estaciones de trabajo sin disco que haya en nuestra red y sistemas de autoconexión.
2) Si deseamos que el espacio disponible en disco se reparta automáticamente entre los diferentes *filesystems*.
3) La contraseña de super usuario.
4) Si deseamos que tras la instalación salgamos a un *shell* o se reinicie automáticamente el sistema.
5) Si estamos de acuerdo con este perfil de instalación, el programa procede a instalar todo el *software* seleccionado en el ordenador, tras lo cual se rearrancará automáticamente, si así se lo hemos indicado.

En este punto, el sistema rearrancará y procederá a instalar el *software* adicional que hayamos seleccionado, eso sí, tenemos que tener en cuenta que esta versión incluye sólo la documentación y el

servidor *Answer Book2*. Una vez realizados estos paso el sistema por fin estará listo para entrar en funcionamiento.

USO Y PRESTACIONES

Sun intenta que *Solaris* resulte sencillo y atractivo de manejar, por ello ha utilizado las librerías *Motif 2.1* y el escritorio *CDE* como base y fundamento de su operatividad en entorno gráfico.

El escritorio *CDE* es un estándar en *Unix* su uso guarda ciertas similitudes con *Windows 3.11*, ya que existe el *Application manager* y un *File manager* de comportamiento similar al del *S.O.* de *Microsoft*. En este escritorio existe además un panel desde el que podemos acceder a determinadas aplicaciones con la ventaja de acceder a menús desplegables, en los que podemos colocar acciones o programas a ejecutar. El escritorio incorpora una serie de aplicaciones para poder trabajar directamente con el equipo

Una de las herramientas que más interesan son los compiladores que trae el sistema. En este caso *Sun* sólo proporciona *Java development kit 1.1.5* y el *kit* de desarrollo *WebNfs* que ofrece acceso *NFS* a las aplicaciones *Java*. Si deseamos desarrollar programas en *C/C++* tenemos dos opciones, la primera es comprar los compiladores a *Sun* y la segunda conseguir una copia del *GNU C* y *C++* para *Solaris* a través de *Internet*, que pueden encontrarse en las direcciones *Internet* de la tabla 2. Ello supone una orientación al desarrollo de aplicaciones *Java*, un lenguaje perfecto para el desarrollo de aplicaciones empresariales en conjunción con potentes motores de bases de datos, como *Oracle*, *Natural Adabas* u otros.

La falta del compilador de C y C++ puede ser subsanada con los compiladores gratuitos disponibles en Internet

De cara a los desarrolladores, *Solaris* viene con una herramienta para el desarrollo cómodo de *GUI*'s (interfaces gráficas de usuario) para las aplicaciones a desarrollar en *C* y que pueden ser utilizados desde otros lenguajes. Genera interfaces compatibles con el escritorio *CDE* que siguen el estándar fijado por él.

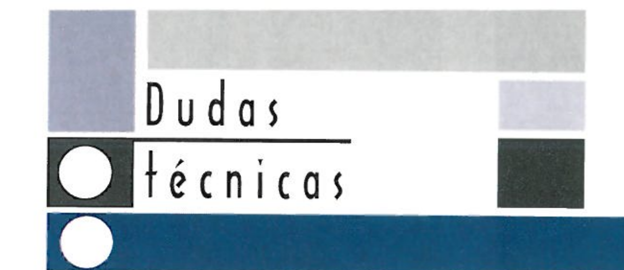
Finalmente, *Solaris* incorpora una herramienta para su configuración denominada *AdminTool* que permite administrar las cuentas de nuestro ordenador, la configuración de los puertos, el *software* que se ha instalado etc. La parte dedicada de la administración de paquetes de *software* es la más interesante, pero en ella apreciamos dos problemas. El primero y más importante es que, una vez que hemos llegado a la selección de paquetes a instalar, el sistema no nos avisa de aquéllos que ya lo están, ni de posibles incompatibilidades, o si no tiene una referencia clara de cuáles tiene ya instalados y cuáles necesita; por el contrario la selección de subpaquetes dentro de los primeros es sencilla y cómoda, permitiendo el agrupamiento jerárquico del *software*.

CONCLUSIÓN

Solaris 7 es un sistema operativo estable y claramente enfocado a la empresa. En su contra pesa la excesiva complicación de su instalación y mantenimiento, en comparación con otros sistemas operativos *Unix* rivales en plataforma *x86*. Por otra parte, su reputación y probada estabilidad hacen que sea un candidato poderoso a la hora de competir por el mercado de los servidores empresariales. Por esto debe considerarse como un *S.O.* que hay que conocer para tener más salidas profesionales aunque esté rezagado en prestaciones respecto de sus rivales más inmediatos.

TABLA-2. Direcciones útiles de Internet para el usuario de Solaris.

URL	Contenido
http://docs.Sun.com	Documentos disponibles de Sun Microsystems
http://www.Sun.com/developers	Programa Developers Connection de Sun
http://www.kempston.net/Solaris/connectanyisp.html	Página sobre el procedimiento de configuración para su conexión a Internet vía módem
http://www.kempston.net/Solaris/	Página de recursos
http://www.sunfreeware.com	Página con software gratuito
http://www.stardivision.com	Página de StarDivision, creadores de StarOffice que esta disponible para Solaris 7



Dudas técnicas

En esta sección, como cada mes, SÓLO PROGRAMADORES os brinda la oportunidad de encontrar respuesta a las dudas que podáis tener en algún tema relacionado con la programación bajo cualquier entorno de desarrollo.

Pregunta

Soy un lector asiduo de vuestra revista y estoy muy interesado en la serie que acaba de comenzar acerca del registro de *Windows*. Me gustaría saber dónde puedo recabar información con el fin de saber qué parámetros de configuración de mi sistema se almacenan en el *Registro*. Por otro lado estoy interesado en programar el registro desde programas que hago yo mismo en *Visual C++*. ¿Qué cambios implica esto con respecto a lo que se cuenta en la serie basada en *Visual Basic*? Finalmente me gustaría saber qué herramientas puedo utilizar para hacer una copia de seguridad del *Registro*.

Muchas gracias.

Respuesta

Querido lector:

Ante todo gracias por tu confianza. Empezaremos por la pregunta más fácil. Aunque la serie sobre el *Registro* de *Windows* está planteada desde el punto de vista de *Visual Basic* en realidad las funciones y procedimientos que se están utilizando proceden de la *API* de *Windows*. Por lo tanto básicamente puedes utilizar las mismas llamadas, eso sí, adaptando su uso a *Visual C++*. En este sentido consulta la ayuda proporcionada por el propio entorno de *Visual C++* y observarás como no existe diferencia formal entre esas llamadas y las utilizadas en los artículos. Simplemente se optó por *Visual Basic* porque es uno de los lenguajes más sencillos de cara al aprendizaje.

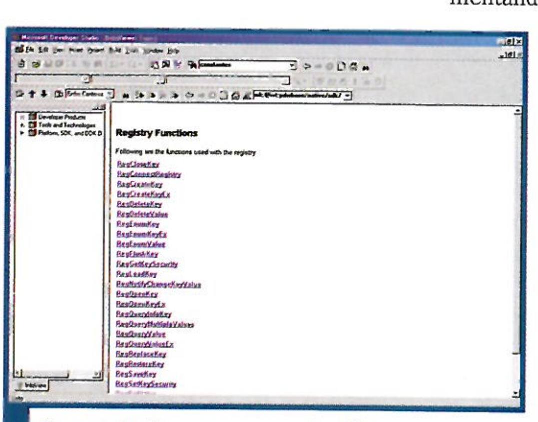


Figura 1. Ayuda que proporciona Visual C++ con respecto a las llamadas para programar el Registro de Windows.

Realmente, como tu mismo apuntas, para programar el *Registro* de *Windows* no es suficiente con saber cuáles son las llamadas de la *API* y cómo usarlas. Hay que conocer el *Registro* en profundidad para saber qué es lo que podemos modificar. En *Internet* encontrarás algunas direcciones muy interesantes, tutoriales y ejemplos. Te recomendamos especialmente el sitio *Web* **http://www.regedit.com.** En él encontrarás información detallada acerca de la estructura interna del *Registro* y su composición. No obstante para que vayas abriendo boca te resumimos brevemente un par de entradas del *Registro* para que puedas ir experimentando por tu cuenta:

- Cómo hacer aparecer y/o desaparecer las flechas que aparecen en los iconos de acceso directo en *Windows*.

```
HKEY_CLASSES_ROOT\lnkfile
HKEY_CLASSES_ROOT\piffile
```

En estas dos entradas aparecerá un valor que es *IsShortCut*. Tendremos que borrarlo de ambas si queremos que desaparezcan las flechas que aparecen sobre los iconos de acceso directo. Si deseamos volver a añadir la flecha sólo tenemos que añadir la clave *IsShortCut*.

- Cómo modificar, borrar o añadir la lista de aplicaciones que aparecen cuando hacemos *clic*

con el botón derecho y elegimos **Nuevo** en el primer menú de contexto de *Windows*.

```
HKEY_CLASSES_ROOT
```

Esta entrada contiene una lista con las distintas extensiones registradas en *Windows*. Una extensión aparecerá en el menú *Nuevo* si dentro de ella existe una referencia que es *ShellNew*. A su vez, dentro de *ShellNew* habrá un valor que es *NullFile*. Borrando la clave *ShellNew* se eliminará la extensión correspondiente de la opción *Nuevo*. Para añadir una nueva opción al menú *Nuevo* será preciso incluir la clave *ShellNew*. Después, dentro de ésta tendremos que incluir los parámetros asociados a esa aplicación. Normalmente suele ser la clave *NullFile* con valor igual a "".

Por último, haces bien en preocuparte por la seguridad, especialmente ahora que vas a programar el *Registro* de *Windows*. Toda precaución es poca. Existen multitud de herramientas que hacen copias de seguridad de elementos vitales del sistema, pero concretamente para el *Registro* te remitimos a las dos propuestas que aparecían en el primer artículo de la serie. Tanto una como la otra son soluciones sencillas y gratuitas que te permitirán tener a buen recaudo el *Registro* del sistema.

Pregunta

Me dirijo a la revista Solo Programadores porque he estado siguiendo atentamente la serie que vienen publicando desde hace unos meses acerca del *XML*. Lo cierto es que sigo sin tener nada claro qué diferencia al *XML* del *HTML*. Tampoco comprendo muy bien cuáles son las ventajas del *XML* y realmente para qué sirve. ¿Se utiliza en realidad? ¿Dónde puedo ver esa tecnología en funcionamiento?

Respuesta

Estimado lector:

No eres el único que se plantea todos estos temas. Intentaremos aclarar lo más sencillamente posible todas tus dudas. El lenguaje *HTML* sirve fundamentalmente para definir cómo han de presentarse los datos, mientras que el lenguaje *XML* se encarga de precisar cómo son los datos. En realidad *HTML* y *XML* no se parecen en nada desde el punto de vista de aquello para lo que sirven, pero su sintaxis es similar y de ahí procede en muchos casos la confusión. Ambos utilizan el sistema de etiquetas. Los navegadores ya llevan mucho tiempo interpretando este tipo de lenguaje por lo que parece lógico que el estándar *XML* se incorpore de manera natural a las aplicaciones *Web*.

conocimiento que ayude a saber qué estructura presenta dicha información, de qué tipo son los elementos contenidos en ella, etc. Aquí es donde el estándar *XML* tiene cabida, ya que embebido dentro de las páginas *HTML* confiere a la información la naturaleza de datos susceptibles de ser manejados.

El estándar *XML* se está extendiendo poco a poco, de manera gradual, por todos los ámbitos de desarrollo de soluciones basadas en Internet. Un ejemplo de esto lo tienes en el conocidísimo servidor de aplicaciones llamado *ColdFusion* (**http://www.allaire.com**). Este ha sido uno de los primeros en ofrecer soporte para la tecnología *XML*. Poco a poco los fabricantes de bases de datos se están sumando y no tardaremos mucho en ver aplicaciones que usen de manera exhaustiva este estándar.

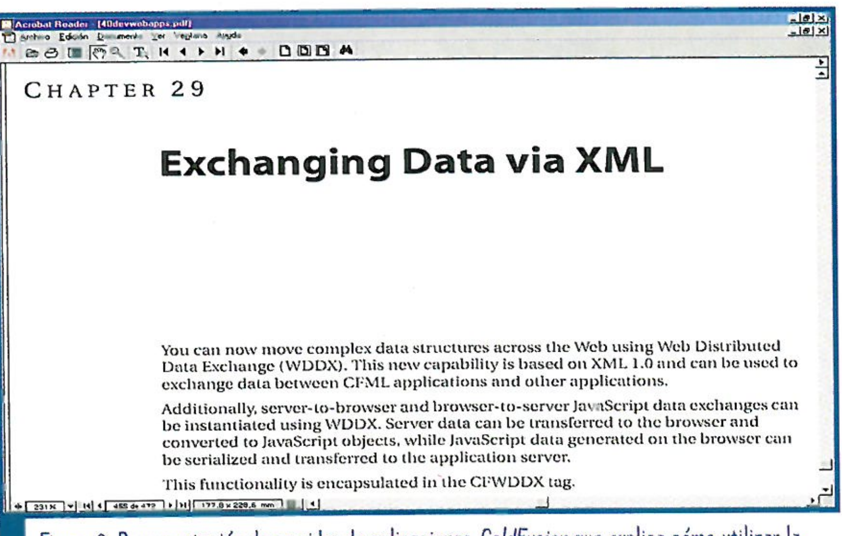


Figura 2. Documentación de servidor de aplicaciones *ColdFusion* que explica cómo utilizar la tecnología *XML* para el intercambio de datos.

Antes de pasar a entender el leguaje propiamente dicho es de vital importancia que comprendas la necesidad y utilidad del estándar *XML*. Los datos no pueden ser tratados como tales hoy en día ya que éstos se entregan normalmente en forma de páginas *HTML*. Las etiquetas *HTML* dictan al navegador cómo ha de mostrar la información pero no proporcionan ningún otro

En cuanto al cliente *Internet Explorer 5.0* proporciona un excelente soporte para *XML*, lo que permite comprobar de manera sencilla las virtudes de este lenguaje. Desde aquí te animamos sinceramente a que te pongas manos a la obra con los ejemplos que aparecen en todos los capítulos de la serie. Estamos seguros de que en no mucho tiempo serás un convencido más.

Pregunta

Estimados amigos de SOLO PROGRAMADORES, os escribo porque he visto hace unos días en *Internet* una página *Web* donde aparecía una lista de selección junto a un campo de texto. Cuando tecleaba en el campo de texto en la lista se seleccionaba la opción más cerca. Me gustaría saber si eso es *HTML Dinámico*, y, sobre todo, cómo se puede hacer.

Muchas gracias.

Respuesta

Querido lector:

El término *HTML Dinámico* se presta a mucha confusión ya que se utiliza con diversos significados. En cualquier caso el pequeño truco del que nos hablas en relación con esa página *Web* no es demasiado complicado. Desde aquí te proponemos una primera y rápida aproximación sobre la que tú, seguramente, podrás hacer variaciones y mejoras.

Supongamos que tenemos el siguiente formulario.

```
<FORM>
Seleccione una
    canci&oacute;n:<BR>
<INPUT TYPE="Text" NAME="CANCION"
    SIZE="30" onkeyup="hacerKey-
    Press()"><BR>
<SELECT NAME="LISTA_CANCIONES"
    SIZE="5">
<OPTION VALUE="1">Drowned
    World/Substitute For Love
<OPTION VALUE="2">Swim
<OPTION VALUE="3">Ray of light
<OPTION VALUE="4">Candy Perfume
    Girl
<OPTION VALUE="5">Skin
<OPTION VALUE="6">Nothing Really
    Matters
<OPTION VALUE="7">Sky Fits Heaven
<OPTION VALUE="8">Shanti/Ashtangi
<OPTION VALUE="9">Frozen
<OPTION VALUE="10">The Power Of
    Good-Bye
<OPTION VALUE="11">To Have Not To
    Hold
<OPTION VALUE="12">Little Star
<OPTION VALUE="13">Merl Girl
</SELECT>
</FORM>
```

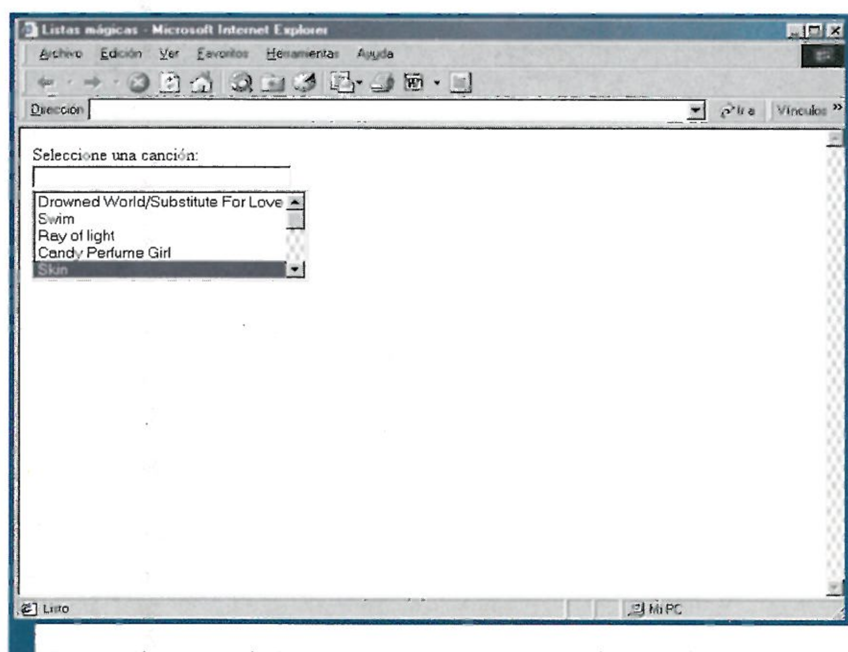


Figura 3. Clásico ejemplo de página con campo de texto junto a lista de selección.

Uno de los muchos eventos que se pueden aplicar sobre el campo de texto es *onkeyup* (al igual que *onkeydown* y *onkeypress*, aunque, dada su naturaleza, el comportamiento de los tres varía ligeramente). En este caso concreto, este evento se dispara cada vez que el usuario teclea algo en el campo de texto. Este es el momento en el que debemos comprobar si lo tecleado coincide con alguno de los elementos de la lista desplegable, sobre todo para evitar los consabidos errores.

```
<SCRIPT LANGUAGE="JavaScript">
<!-
function hacerKeyPress() {
    var i, cadena;
    var f = document.forms[0];

    for (i=0;i<f.LISTA_CANCIO-
    NES.length;i++) {
        cadena =
        f.LISTA_CANCIONES.
        options[i].text;
    if (cadena.indexOf
    (f.CANCION.value) == 0) {
        f.LISTA_CANCIONES.
        selectedIndex = i;
            break;
        }
    }
}
//->
</SCRIPT>
```

En primer lugar recorremos toda la lista de las opciones contenidas en el desplegable. La referencia f.LISTA_CANCIONES.options[i].text contendrá, en cada caso, el texto de la opción correspondiente, por eso lo repetimos con cada una de las opciones elegidas. Por su parte el método *indexOf* devuelve la posición en la que aparece una determinada cadena dentro de otra. Por consiguiente, si esa posición es cero estamos ante un comienzo de palabra similar y por lo tanto la propiedad *selectedIndex* de la lista desplegable se actualiza con el valor de índice del bucle en ese momento.

TODOS LOS MESES EN TU QUIOSCO



PARA NO QUEDARSE HELADO CUANDO HABLEN DE LINUX





PARA ESTAR AL DÍA EN SOFTWARE SIN GASTARSE EN TELÉFONO UN EURO

- LOS MEJORES ESPECIALES
- DISEÑO
- INTERNET
- MULTIMEDIA
- UTILIDADES
- EN CASTELLANO
- NEGOCIOS
- LO ÚLTIMO
- Y MUCHO MÁS...

LOS PROFESIONALES EXIGENTES SÓLO NECESITAN QUE LES RECORDEMOS UNA FECHA

1 2 3 4 5 6 7 Noviembre

PROFESIONALES

PÚBLICO EN GENERAL



Feria Internacional de Informática,
Multimedia y Comunicaciones

Madrid, 2-7 Noviembre de 1999

